DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A PRODUCTIVITY ENHANCEMENT STUDY FOR THE
U.S. ARMY INFORMATION SYSTEMS
ENGINEERING COMMAND

by

Timothy F. Robertson

September 1985

Thesis Advisor: Norman R. Lyons
Co-Advisor: David Whipple

Approved for public release; distribution is unlimited

T226811

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
A Productivity Enhancement Study for the U.S. Army Information Systems Engineering Command

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis
September 1985

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Timothy F. Robertson

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, CA 93943-5100

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, CA 93943-5100

12. REPORT DATE
September 1985

13. NUMBER OF PAGES
134

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
productivity, productivity improvement program, software development, software maintenance, software tools, software development environment, prototyping, evolutionary development

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
A productivity enhancement study for the U.S. Army Information Systems Engineering Command (ISEC) is described. Recommendations for improvement and recommendations for further study are provided. ISEC has an important mission with regard to managing the Army's information resources. ISEC is tasked with developing and maintaining the Standard Army Multi-command Management Information Systems (STAMMIS). Because of resource constraints and increased mission requirements, it is essential that ISEC (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

ABSTRACT (Continued)

increase productivity to meet the information needs of the Army. Specifically, this thesis: (1) evaluates the traditional software life cycle in contrast with prototyping and evolutionary development; (2) discusses project management issues; (3) explains the need for integrated software tools; (4) discusses human factors in the software development process; and (5) proposes a system for capturing and measuring productivity.

Approved for public release; distribution is unlimited.

A Productivity Enhancement Study for the U.S. Army Information Systems Engineering Command

by

Timothy F. Robertson
Captain, United States Army
B.S. Arizona State University

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 1985

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

ABSTRACT

A productivity enhancement study for the U.S. Army Information Systems Engineering Command (ISEC) is described. Recommendations for improvement and recommendations for further study are provided. ISEC has an important mission with regard to managing the Army's information resources. ISEC is tasked with developing and maintaining the Standard Army Multi-command Management Information Systems (STAMMIS). Because of resource constraints and increased mission requirements, it is essential that ISEC increase productivity to meet the information needs of the Army.

Specifically, this thesis: (1) evaluates the traditional software life cycle in contrast with prototyping and evolutionary development; (2) discusses project management issues; (3) explains the need for integrated software tools; (4) discusses human factors in the software development process; and (5) proposes a system for capturing and measuring productivity.

TABLE OF CONTENTS

I. INTRODUCTION 10

A. PROBLEMS WITH SOFTWARE DEVELOPMENT 10

1. Reduced Hardware Costs 11

2. Applications and Invisible Backlogs . . . 11

3. Shortage of Software Personnel 11

4. Problems with the Specification Process 12

5. Perception Problems 12

6. Abstract Product 12

7. Technological Change 13

8. Government Laws, Regulations, Policy and Traditions 13

B. THESIS ORGANIZATION AND OBJECTIVES 15

II. SOFTWARE DEVELOPMENT AT ISEC 17

A. ARMY INFORMATION MANAGEMENT AND ISEC 17

1. Recent Historical Background 17

2. Introduction to ISEC 18

3. The Evolution of Army Information Management 19

B. THE STAMMIS DEVELOPERS 21

C. BASIC PROBLEMS AT ISEC 23

1. Preface 23

2. Specification Problems 23

3. Political Conflicts 24

4. Lack of Skilled Professionals 24

5. Personnel Reductions 25

6. Contract Management Problems 25

7. Lack of Standardization 27

8. Organizational Turbulence 27

9. File Processing Shortcomings 27
D. CONCLUSION 29

III. PRODUCTIVITY MEASUREMENT AND IMPROVEMENT 30
A. INTRODUCTION 30
1. Overview 30
2. What Productivity is Not 30
3. Productivity and Software Development/Maintenance 31
B. MEASUREMENT FOR IMPROVEMENT 32
1. Why We Should Measure Productivity 32
2. Units of Measurement 34
C. IMPLEMENTATION OF A PRODUCTIVITY MEASUREMENT SYSTEM 47
1. Overview 47
2. Implementing the Measurement System . . . 47
D. SUMMARY 50

IV. REQUIREMENTS ANALYSIS AND SYSTEMS DEVELOPMENT WITH PROTOTYPING 52
A. PROBLEMS WITH THE TRADITIONAL SOFTWARE LIFE CYCLE 52
1. The Traditional Software Life Cycle . . . 52
2. Maintenance Considerations 58
B. PROTOTYPING AND EVOLUTIONARY DEVELOPMENT . . . 61
C. ADVANTAGES OF PROTOTYPING 63
D. PROTOTYPING LIMITATIONS 65
E. A BRIEF OVERVIEW OF THE PROTOTYPING LIFE CYCLE 67
F. PROTOTYPING APPLICABILITY AND IMPLEMENTATION AT ISEC 68
G. PROTOTYPING AND EVOLUTIONARY DEVEOPMENT: THE BOTTOM LINE 72

V. THE SOFTWARE DEVELOPMENT ENVIRONMENT 74
A. INTRODUCTION 74
B. THE PHYSICAL ENVIRONMENT 75
C. THE STRUCTURAL ENVIRONMENT 78

1. Staffing 77
2. Awards and Incentives 79
3. Suggestion Programs 80
4. Flextime 81
D. DEVELOPMENT AND MAINTENANCE TOOLS 83
1. Hardware Considerations 83
2. Software Tools 85
E. THE "INTELLECTUAL SKILLS" ENVIRONMENT 92
F. SUMMARY 93

VI. SOFTWARE MANAGEMENT AND PRODUCTIVITY ISSUES . . . 95
A. INTRODUCTION 95
B. GENERAL MANAGEMENT ISSUES 95
1. Management Barriers to Productivity . . . 95
2. Close to the Customer 99
3. System Quality and Productivity 101
C. SOFTWARE DEVELOPMENT CONTRACTS 102
D. PROJECT PLANNING SYSTEMS 104
1. Preface 104
2. Project Management Software 106
3. The Putnam SLIM System 108
4. The COCOMO System 109
E. PRODUCTIVITY IMPROVEMENT PROGRAM 110
F. SUMMARY 115

VII. CONCLUSIONS AND RECOMMENDATIONS 118
A. THESIS SUMMARY 118
B. RECOMMENDATIONS 119
C. RECOMMENDATIONS FOR FURTHER STUDY 121

APPENDIX A: ACRONYMS 123

APPENDIX B: USING GRADUATE STUDENTS TO IMPROVE PRODUCTIVITY 126

LIST OF REFERENCES 128

INITIAL DISTRIBUTION LIST 134

LIST OF TABLES

I	STAGES OF DP GROWTH - NOLAN	20
II	ADVANTAGES OF DATA BASE PROCESSING	28
III	POTENTIAL BENEFITS OF MEASURING PRODUCTIVITY	34
IV	ADVANTAGES OF THE 3 BASIC TYPES OF PRODUCTIVITY MEASURES	36
V	LIMITATIONS OF THE 3 BASIC TYPES OF PRODUCTIVITY MEASURES	37
VI	PROBLEMS WITH LINES OF CODE AS A PRODUCTIVITY MEASURE	41
VII	PROBLEMS WITH THE SPECIFICATION PROCESS	56
VIII	ADVANTAGES OF PROTOTYPING	64
IX	LIMITATIONS OF PROTOTYPING	66
X	THE PHYSICAL PROGRAMMER ENVIRONMENT - IBM	76
XI	STAFFING PRINCIPLES - BOEHM	79
XII	BENEFITS OF PQ TEAMS - SUMANTH	83
XIII	BENEFITS OF THE USE.IT TOOL SYSTEM	91
XIV	SOFTWARE DEVELOPMENT CONTRACT PROBLEMS IN THE GOVERNMENT	104
XV	GUIDELINES FOR SOFTWARE DEVELOPMENT CONTRACTS	106
XVI	KEY POINTS - SOFTWARE PRODUCTIVITY IMPROVEMENT PROGRAM	116
XVII	PRODUCTIVITY IMPROVEMENT PROGRAM PROBLEMS AND SOLUTIONS	117

LIST OF FIGURES

2.1 STAMMIS Functional Areas, Examples and Proponents 22
4.1 The Traditional Software Life Cycle 53
4.2 The Department of Defense System Life Cycle 54
4.3 Relative Cost to Correct Errors in Software Life Cycle 57
4.4 Maintenance - The Largest Cost Driver 59
4.5 Relative Effort for Maintenance Activities 60
4.6 The Prototyping Life Cycle 67

I. INTRODUCTION

A. PROBLEMS WITH SOFTWARE DEVELOPMENT

This study reviews how the U.S. Army Information Systems Engineering Command (ISEC) develops and maintains the software for the Army's management information systems (MIS). The purpose of this thesis is to recommend ways of improving productivity throughout STAMMIS development and maintenance while maintaining or improving quality. Resource constraints, budget and manpower reductions, and expanding mission requirements have put pressure on ISEC to look for better, faster, and smarter ways of doing business. Adding to the difficulty is a whole host of software development problems that plague the government and most private companies as well.

The computer-related literature is replete with horror stories about software products that have failed in one respect or another. Software developers have earned a bad reputation for delivering products that do not meet user requirements, are late and are over budget. What are the factors that are the roots of the so called "software crisis?" The major contributing factors are: (1) reduced hardware costs; (2) the applications backlog; (3) a shortage of skilled software personnel; (4) the difficulties of specifying what the software should do; (5) user perception problems; (6) the abstract nature of the product; (7) the rapid pace of technological change; and (8) curious governmental regulations and policies.

1. Reduced Hardware Costs

The cost of computer hardware has been dropping significantly during past several years. The average cost decrease per year has been 15-30 percent. At the same time, the processing power of computers has risen dramatically. The resulting cost/performance ratio has improved immensely. From 1959 to 1979, for example, it improved by six orders of magnitude. To put this in perspective, a unit of processing power that cost $1,000,000 in 1959 would cost only one dollar in 1979. [Ref. 1: p. 84] Meanwhile, labor costs have continued to rise. Today, software costs typically represent the largest expenditure for systems.

2. Applications and Invisible Backlogs

Government leaders and industry management are realizing that information is a resource. It requires management, security, planning, and control just like other precious assets. Users are becoming more sophisticated. There is more hardware to support. These factors have fueled demand for new applications at a rate faster than present programmers can provide them. Many organizations have a backlog of programming projects ranging from two to four years. Behind this documented backlog of projects is often an equally large "invisible backlog" of user requirements that are unfulfilled. No one prepares the justification to document these needs because there is no hope of getting results in any reasonable period of time. [Ref. 2: pp. 281-284]

3. Shortage of Software Personnel

There is an acknowledged shortage of skilled programmers, analysts, and software managers. Estimates in 1984 reflected a shortage of software personnel in the U.S.

of almost 100,000 with the gap expected to widen in the near term. [Ref. 3: p. 30] The competition for professional software personnel is intense in the private sector. Government agencies have a difficult time competing with industry for skilled personnel under these conditions.

4. Problems with the Specification Process

In software development, the key to writing good software is capturing accurate and complete specifications. The user and developer are usually from different technical backgrounds. Thus, there is a natural cultural communications gap between them. Usually the user does not know exactly what he wants. Changes are common as more experience is gained building and using the system. The traditional software life cycle forces the developer to freeze the specifications so that detailed design and coding can begin. This results in an unstable foundation on which to build the software. [Ref. 4: pp. 59-61]

5. Perception Problems

Most people neither understand nor appreciate the problems involved in developing or maintaining software. It is much more difficult to build software than our intuition tells us that it should be. To the user, simple changes appear to take much too long to implement. He gets frustrated which leads to polarization between "the DPers" and other organizational elements.

6. Abstract Product

Except for coding and documentation which are typically done late in the software life cycle, there are few tangible products by which to measure progress. Lack of a physical product makes scheduling and resource estimation difficult. It makes project management risky business.

Many organizations must also contract out for their software applications because they lack the in-house assets or the expertise to do it themselves. The effective writing and management of software development contracts takes special expertise and experience which many organizations lack.

7. Technological Change

The growth of the computer industry and rapid rate of technological change is unparalleled in history. Managers and technicians alike have difficulty staying on top of the latest developments in the field. Some react by burying their heads in the sand and resisting all change. Others attempt to solve the wrong problem by only trying to improve traditional methods. If we are trying to actually improve productivity, we must look for new and innovative ways to solve problems. In this regard, automation should be used to the maximum extent possible. [Ref. 2: pp. 11-14]

8. Government Laws, Regulations, Policy and Traditions

Another class of problems exists for ADP organizations in the government. The laws and regulations developed when computers (hardware) were extremely expensive are still on the books today. The primary bill governing acquisition of computer equipment is the Brooks Act (Public Law 89-306). The effect of the Brooks Act has been to create layers of administrative actions required to justify and procure new hardware. With the incredible improvement in the cost/performance ratio of computers, legislation of this nature represents a double barrier to productivity. Not only does it tie up manpower preparing and staffing the necessary documentation to justify the procurement, but the benefits of the new technology or methodology are foregone. In addition, the government is not able to take advantage of the bargains available due to the current economic slump.

Certain government policies can be counter-productive. A striking example of this is Department of Defense (DOD) Directive 5000.29, entitled "Management of Computer Resources in Major Defense Systems." It requires the use of a DOD-approved higher order language in defense systems unless it can be proven that another language would be more cost effective for a specific application.[1] The intent was to stem the tide of language proliferation. [Ref. 5: p. 15] What it did, however, was close the door on fourth generation languages which were in their infancy at the time (1976).

Current civilian personnel office (CPO) policies do not normally allow technician positions in the grade of GS-13 and above. Those are strictly for management positions. This promotes the Peter Principle. Some technicians are promoted who really do not want to be managers or lack the requisite skills. Others stay in their GS-12 positions but develop morale problems which lead to decreased efficiency and productivity. For those technicians who leave for private industry, it means the government incurs increased recruitment costs, increased training costs, loss of institutional knowledge, learning curve productivity losses, and personnel vacancies. Computer technology is complex and changing so rapidly that skilled technicians at senior levels are not a luxury but a necessity.

Recent budget cutting schemes have not helped recruiting or retention. Announcing a potential five percent wage reduction for fiscal year 1986 degraded morale of those in the work force. For those considering work in

---

[1]DOD Directive 5000.31 provides the actual interim list of approved languages. Both directives were issued in 1976, before the advent of true fourth generation languages.

the public sector, wage reductions add one more strike against government employment.

The work environment for governmental agencies is often austere. For many software development agencies, the work space is cramped and there are too few conference rooms for meetings and walkthroughs. Additionally, the programmer to terminal ratio is poor, and the programmers complain of poor computer response time. Software development tools, if available, are dated and the programmers lack training in how to tap their full potential.

B. THESIS ORGANIZATION AND OBJECTIVES

The introduction has provided background information on software development problems which are applicable to nearly all governmental and private organizations. It will serve as a framework from which to discuss problems and specific productivity issues at ISEC. The U.S. Army Information Systems Engineering Command faces significant challenges in the near and distant future. To meet their mission requirements, internal goals, and objectives, they must improve productivity.

Chapter two provides an organizational overview of ISEC and its role in information resource management in the Army. The chapter will discuss Army MIS and the role of each major player in the process. It will conclude by discussing specific productivity problems at ISEC.

Chapter three looks at productivity measurement. Several methods for measuring productivity will be discussed. Some metrics are needed to determine whether productivity is increasing, decreasing or unchanged. Productivity measurement in the software field is neither free nor easy. The productivity yardsticks chosen must be carefully selected to avoid influencing employee behavior

that would have a negative impact to the organization as a whole.

Chapter four will discuss the traditional software life cycle and the inherent problems associated with developing software using that methodology. The role of requirements definition is absolutely critical for developing software that meets users needs in a timely fashion. For reasons we will discuss in chapter three, the most expensive problems in developing software have historically been in the requirements definition phase. An alternate life cycle using prototyping is proposed. Prototyping is a departure from the traditional software life cycle. The advantages and limitations of prototyping will be addressed as well as management implications and issues involved in adopting such a life cycle.

In chapter five, we will explore the software development environment which includes software tools, techniques, and the technologies employed developing the software. The need for an integrated software tool set to support software development and maintenance will be established. This chapter will also discuss human factors and how they relate to productivity. The human factors considered will include motivation, awards and incentives, employee training, working conditions, and computer response time.

Chapter six is concerned with software management. We will discuss project planning and project management, resource estimation, contract management, and some general management issues. A formal productivity improvement program is suggested.

The final chapter will present conclusions reached during this study and summarize recommendations for improving productivity. It will also recommend areas that require further study.

II. SOFTWARE DEVELOPMENT AT ISEC

A. ARMY INFORMATION MANAGEMENT AND ISEC

1. Recent Historical Background

In 1984, a major reorganization in the Army hierarchy occurred which will have profound strategic implications in the years to follow.

> On May 9, 1984, General John A. Wickham Jr., chief of staff of the Army, took the first steps to improve information management. He approved the establishment of the information mission area (IMA). This decision brought together and integrated the subfunctions of IMA - telecommunications, automation to include office automation, audiovisual, records management and publications. [Ref. 6: pp. 30-33]

Among the changes in the Army's organization included the addition of a fifth arm of the Department of the Army (DA) general staff, the Assistant Chief of Staff, Information Management (ACSIM). The mission of ACSIM is:

> to improve the management quality and flow of information as a principal resource in achieving total Army goals, by fully integrating all information functions, including information resource management, communications, administration and command and control. [Ref. 7: p. 4]

Another change resulting from General Wickham's IMA guidance was the establishment of the U.S. Army Information Systems Command (USAISC). USAISC operates and maintains assigned information systems in the area of telecommunications, automation, office automation, and audiovisual [Ref. 8: p. 35]. USAISC was created from assets of the U.S. Army Communications Command (USACC), the U.S. Army Computer Systems Command (CSC) and several other smaller units.

The 1984 reorganization also created the Information Systems Software Support Command (ISSSC), the forerunner to ISEC. It was formed from the remaining assets of the Computer Systems Command and placed under the leadership of USAISC. In June 1985, ISEC was formed by merging the assets of the ISSSC and U.S. Army Information Systems Software Support Command (ISSSC) and the U.S. Army Electronics System Engineering and Installation Command (AESEIC). AESEIC was stationed at Fort Huachuca, Arizona, and most of its functions and employees will remain there but under the auspices of ISEC. Despite the name change and merger, ISEC remains a subordinate unit of USAISC today.

2. Introduction to ISEC

The U.S. Army Information Systems Engineering Command (ISEC) is a key organization in the Army's effort to manage its vast information resources. ISEC has responsibility for the technical aspects involved in developing, designing, testing, implementing and maintaining the Army's Standard Army Multi-command Management Information Systems (STAMMIS).

The command is headquartered at Fort Belvoir, Virginia, about 12 miles south of Washington D.C. Several hundred employees work in nearby Falls Church, Virginia while a major programming directorate is located at Fort Lee, Virginia. In addition, ISEC has operational units located around the globe with support teams in Hawaii, Germany, and Korea. More than 1500 hundred ISEC employees (former AESEIC employees) work at Fort Huachuca, Arizona, but their primary duties are not STAMMIS related.

A considerable portion of ISEC resources are involved in STAMMIS development and maintenance. Their other missions include but are not limited to:

1. Managing the Army Information Processing Standards (AIPS) program;
2. Providing technical support to all Army echelons;
3. Conducting software research;
4. Developing software standards;
5. Serving as a developer and evaluator of systems software and executive software; and
6. Monitoring hardware development.

The size and responsibilities placed on ISEC are impressive. It is responsible for information systems design, development, installation and test to include hardware, software, and systems integration. [Ref. 9: p. 38] ISEC employs well over 4000 workers most of whom are civilians. It is comparable in size to major software houses and computer companies. Approximately 1000 employees are in programmer or analyst positions supporting STAMMIS.

3. The Evolution of Army Information Management

During the past 20 years, the Army has modified its information structure several times. As the Army's information needs evolved and computer-based information systems technology advanced, the Army's philosophy for managing information has also evolved. The IMA reorganization demonstrates a commitment by the Army to treat information as one of its most precious resources. The consolidation and integration of information functions under one manager makes sense because of the interrelationships between them.

Richard Nolan wrote a landmark article in the Harvard Business Review (1979) on the stages of evolutionary growth that organizations experience with data processing. Nolan describes how organizations move through rather distinct stages from stage 1 (initiation) through stage 6

TABLE I

STAGES OF DP GROWTH - NOLAN

STAGE 1: INITIATION

* Development of low level operational systems in a functional area
* No overall planning or control

STAGE 2: CONTAGION

* Growing demand for and proliferation of applications
* Innovation encouraged
* Applications developed in isolation
* Low level of planning and control
* Managers cannot obtain information for decision making
* Proliferation of incompatible and redundant data

STAGE 3: CONTROL

* Planning and control become formalized
* Shift occurs in management orientation from management of computers to management of the company's data resources
* Users arbitrarily held accountable for the cost of data processing; Users become frustrated

STAGE 4: INTEGRATION

* Existing applications retrofited into data bases
* Increased demand by users
* Redundancy of data

STAGE 5: DATA ADMINISTRATION

* Organization wide strategic planning
* IRM emphasized
* Tailored planning and control systems

STAGE 6: MATURITY

* Applications portfolio is completed
* Structure mirrors the enterprise and the information flow in the company
* Information Engineering is largely completed

(maturity).[2] [Ref. 10: pp. 115-126] These stages differ in the kinds of applications being developed, the control over the information system function, and by the degree of planning involved for future applications. [Ref. 11: pp. 269-72] Table I is a summary of Nolan's six stage model. It is normal for an enterprise to reorganize (such as the Army's evolution) as it attempts to control and make full use of its information resources.

Where does the Army fit in Nolan's Model? Although the boundaries are somewhat ill-defined, the Army is somewhere between stages 3 and 5. Strategic planning and Information Resource Management (IRM) are being emphasisized, thus, one could argue that the Army is beginning data administration (stage 5). On the other hand, a strong case could be made that the Army has never left stage 3. Users are frustrated with the applications backlog and management has frequent difficulty obtaining the information they desire for decision making. Additionally, the Army has not retrofited existing applications into data bases.

B. THE STAMMIS DEVELOPERS

There are three main participants involved in the development of STAMMIS: the functional proponent (FP), ISEC, and the user. The functional proponent is normally a Department of the Army (DA) staff element such as Deputy Chief of Staff, Personnel (DCSPER). The FP is responsible for the functional software specifications of a particular STAMMIS. They also assist with functional aspects concerning STAMMIS design, development and testing. ISEC is

---

[2]This is a refinement of an earlier article by Gibson, Cyrus F., and Nolan, Richard L., "Managing the Four Stages of EDP Growth," Harvard Business Review, v. 52, January--February 1974, pp. 66-76.

responsible for the technical specification, design, development, coding, configuration management,testing, implementation, and maintenance of the STAMMIS. ISEC is known as the application system developer (ASD) for STAMMIS Software. [Ref. 12: p. A-1]

The Army's STAMMIS or management information systems (MIS) can be divided into three main functional areas. They are shown in figure 2.1 along with the functional proponent and examples of each. Each functional area is serviced by a separate ASD within ISEC. In total, ISEC has developed and supports over 60 different MIS.

The sum of all the programs that comprise the more than 60 STAMMIS are on the order of 15-20 million lines of code. This library has been in development for about 20 years. If industrial yardsticks apply, ISEC has spent between one and two billion dollars developing and maintaining this code. [Ref. 13: p. 1]

Functional Area/Example	Functional Proponent
1. Financial Systems	Comptroller of the Army (COA)
STANFINS	
STARCIPS	
2. Logistical Systems	DCSLOG
SAILS	
SARSS	
ULLS	
3. Personnel and Force Accounting Systems	DCSPER
SIDPERS	
VFDMIS	
VTAADS	

Figure 2.1 STAMMIS Functional Areas, Examples and Proponents.

C. BASIC PROBLEMS AT ISEC

1. Preface

It is the opinion of the author that Information Systems Engineering Command is well managed. Employee morale is favorable. ISEC has a sound training program. They are fortunate to have a commander who is technically qualified and understands the issues and problems in systems development and integration. However, in the software development business, the challenges are formidable, varied, and many.

The problems that beset ISEC occur in most other information systems departments or software development organizations. Chapter one considered generic software development problems - all are evident to some degree at ISEC. There are additional factors bearing on the software development efforts at ISEC. They are discussed in the following subsections. Although the role of information is changing in the eyes of Army leadership, there are many obstacles to overcome before achieving Army goals for information resource management.

2. Specification Problems

As mentioned earlier, the key to quality software is capturing accurate and complete specifications. For STAMMIS software, the functional proponent is responsible for the functional specifications. In the unlikely event that the FP knew exactly what it wanted, they often lack the training to write clear, complete specifications. The geographical distance between the STAMMIS key players slows coordination, fosters cultural differences, and increases misunderstandings and development time. In addition, the end user has historically been only marginally involved in the specification process.

STAMMIS have been fielded which did not meet customer needs, were not used, or were difficult to use. There are, indeed, recent examples of just this happening. In 1985, a STAMMIS was developed for the military police (named MPMIS) after considerable pressure was applied by the FP to meet specific deadlines. An audit followup was conducted four months after the system was fielded. The findings revealed that virtually no one was using the MPMIS. [Ref. 14] This has frustrated all parties involved, has had a negative effect on peoples' perceptions of ISEC and dampened ISEC employee morale.

3. Political Conflicts

The FP sometimes makes arbitrary decisions when requesting engineering change packages (ECPs) without consulting ISEC for a resource and time estimates. Because STAMMIS are file processing systems (vice data base processing systems), it is not uncommon for an apparent small change to actually be a time consuming venture. This is due to the ripple effect the change has on other portions of the program or system. This has caused further alienation for two reasons. First, it frustrates the programmers who are forced to work overtime to meet arbitrary deadlines. Second, if a deadline should be threatened, the FP is angry for what seems like the incompetence of ISEC to handle the smallest of changes.

4. Lack of Skilled Professionals

ISEC has a difficult time competing for computer programmers and analysts in the Washington D.C. area. They rarely are able to hire college graduates, much less, computer science graduates. Employees at ISEC give two primary reasons for this: (1) the government does not pay competitive wages to attract these people, and (2) lack of

aggressive recruitment by the servicing CPO. There is evidence in the literature to support the claim that government wages may be lagging behind that of the private sector in the Washington D.C. area. [Ref. 15: pp. 58-69] It should be noted that compensation packages are frequently bundled quite differently making direct comparison of wages dangerous, at best. [Ref. 16: pp. 27-38]

ISEC is forced to "grow their own" and they do just that through an intern training program. Frequently, after their obligation to the government is completed, they leave for better paying jobs in the private sector. Some are hired by software houses or businesses that do contracting work for the Army. These firms pay them a few thousand dollars more which eventually gets charged back to the Army along with a standard overhead markup of 100-150 per cent.

5. Personnel Reductions

The Army is undergoing the process of fielding four active duty light infantry divisions from internal assets to meet worldwide threat scenarios. Despite this buildup, Army leadership (perhaps for good reason) chose not to request an increase to the Army's end-strength from Congress. Organizations such as ISEC are frequently called on to be the "bill payers" for manning such units. The result is that ISEC is asked to do more with less. Several managers interviewed expressed concern over these reductions. They survive through a positive "can do" approach to their work. But at some point, if it has not already been reached, ISEC's ability to meet mission requirements will be threatened.

6. Contract Management Problems

Manpower reductions coupled with the low experience level of the programmers and the demands placed on ISEC by

the functional proponents has created a sizeable backlog of projects. It has forced management to contract out for many projects. This, in and of itself, may not be bad. There are those who argue that the provisions of Office of Management and Budget (OMB) Circular A-76 mandate more STAMMIS work being contracted out. The problem with this, however, is that ISEC has not exhibited the expertise to properly write and manage such contracts.

The Vertical Force Development Management Information System (VFDMIS) contract is a classic "how not to do things" and "if something can go wrong, it will." VFDMIS is a complex STAMMIS expected to be in the neighborhood of one million lines of code (LOC) - no small venture for the finest of software developers. Work on VFDMIS began in 1974. The contract was let to a small business contractor, ASG, despite the knowledge that VFDMIS was to be the largest and arguably the most complex MIS in the Army. The contractor originally lacked the necessary expertise for the project and was hampered by tremendous personnel turnover. In short, VFDMIS has suffered of a history of disappointments, technical difficulties, and problems.

Incredible as it may seem, there have been no deliverables presented to ISEC at the end of any contract year. If the contract was cancelled tomorrow, the Army would have little to show for this 11 year, multi-million dollar fiasco. It doesn't appear likely that the Army will have an operational system for at least a few more years. Coding on the system has only recently begun. [Ref. 17] The chances appear good that the Army will field another obsolete system with obsolete technology.

7. Lack of Standardization

ISEC is required to develop and maintain several versions of a STAMMIS. This is caused by multiple environments in which the software must run. If the Army could agree on a standard operating system such as Multiple Virtual System (MVS), an IBM operating system, the potential long-run savings would be significant. There would be some initial hardware and training costs involved. Some field commanders may not wish to accept the short-run productivity loss that such a change would entail. Further, by standardizing the operating system to MVS, it may stimulate complaints about locking yourself into only a handful of vendors. This is a difficult issue, however, and one that ISEC is exploring.

8. Organizational Turbulence

The initial section of this chapter described ISEC's two major reorganizations in the past 12 months. These changes exact a toll on the productivity of the employees which is difficult to precisely gauge. Adjusting to new relationships and incurring new responsibilities are a part of this transition process. These organizational changes should be of long-run benefit to the Army but there is large payment required today in terms of short-term productivity losses.

9. File Processing Shortcomings

Traditionally, STAMMIS are "stovepipe file processing systems" that are application specific along functional lines. Current STAMMIS fail to take advantage of data base technology. The advantages of data base processing are well known; Table II is a summary those advantages. These advantages may not be fully experienced

in every system. In fact, there are shortcomings of data base processing. Among them are: (1) DBMS systems are expensive; (2) they are complex; (3) backup and recovery are more difficult; and (4) there is an increased vulnerability for failure. [Ref. 18: pp. 1-7]

TABLE II

ADVANTAGES OF DATA BASE PROCESSING

- More information can be obtained from the same amount of data.
- New requests and one-of-a-kind requests are more easily implemented.
- Reduction of data duplication resulting in fewer cases of conflicting reports.
- Program/data independence is achieved thus the data is compatible for other programs.
- Data management is facilitated.
- DBMS generally allow more affordable sophisticated programming.

ISEC has several forward-thinking employees who realize the benefits of data base processing. ISEC have sent a formal request to USAISC to establish a STAMMIS corporate data base [Ref. 19]. USAISC believes the idea has merit but wants ISEC to tie their efforts to the Army corporate data base, an ACSIM initiative [Ref. 20]. This is probably a reasonable idea but because of the lead times involved in designing these systems, it means the current STAMMIS will continue to be file processing systems for several years.

D. CONCLUSION

The pitfalls to software development are staggering but not unsolvable. In this regard, there are four basic approaches to improving productivity at ISEC. These involve:

1. Improving the techniques and methodologies employed developing the STAMMIS (Chapter 4);
2. Utilizing the benefits of technology to develop the system more effectively and efficiently (Chapter 5);
3. Creating a positive work environment for the employees at ISEC (Chapter 5); and
4. Improving the Management of STAMMIS (Chapter 6).

The next chapter defines productivity and explains productivity measurement in general. It discusses specific productivity measurements for software development and maintenance. It also discusses implementing a performance measurement system at ISEC. The remaining chapters discuss the four basic approaches mentioned above to provide suggestions for improving productivity while simultaneously improving (or at least maintaining) the overall quality of the STAMMIS.

III. PRODUCTIVITY MEASUREMENT AND IMPROVEMENT

A. INTRODUCTION

1. Overview

Productivity is a favorite buzzword of the 1980s. Data processing literature is laced with articles on different aspects of the subject. The acute programmer shortage and the great demand for software products have made productivity an especially critical issue in software development. Published goals at ISEC state "Foster Productivity" as a principal organizational objective. What is productivity? How do you foster productivity? These questions and related issues are discussed in this chapter.

2. What Productivity is Not

Productivity is often confused with production. It does not necessarily follow that the greater the production, the greater the productivity. Surprisingly, a 1972 Louis Harris poll revealed that 27 percent of executives interviewed held this erroneous view. In addition, one third of all college educated and professional respondents were likewise ill-informed. [Ref. 21: p. 41] The following description should help clarify the distinction between the two:

> Production is concerned with the activity of producing goods and/or services.
>
> Productivity is concerned with the efficient utilization of resources (inputs) in producing goods and or services (outputs). [Ref. 22: p. 4]

There is also confusion between the related concepts of efficiency, effectiveness, and productivity. Effectiveness reflects how well a result met its objectives; efficiency reflects how well the resources are utilized to accomplish the results. Thus productivity is really a combination of both effectiveness and efficiency since effectiveness is related to performance while efficiency is related to resource utilization. [Ref. 22: p. 6]

3. Productivity and Software Development/Maintenance

We have talked about productivity throughout this paper but have yet to define it. Alvin Toffler, author of The Third Wave, discusses the problems of defining productivity:

> The first problem is the definition of productivity. It is one of the spongiest, and one of the most treacherous of economic concepts. It was designed for a world of material production, when you could count how many workers and how many hours it took to turn out how many skirts or copper bars. As we have moved to what I've been calling a Third Wave economy, more and more of our output consists of information, services, experience. More and more the consumers' own actions affect the efficiency of the producer. In addition, we have begun to appreciate that economic productivity is frequently more an artifact of accounting and of permissible externalization than of anything else. So I have tremendous problems with the very term "productivity." [Ref. 23: p. 14]

Perhaps turning to the dictionary will serve as a starting point. Webster's Third New International Dictionary defines productivity as:

a) The physical output per unit of production effort;
b) The degree of effectiveness of industrial management in utilizing the facilities of production; especially the effectiveness in utilizing labor and equipment.

Dr Irving Siegel, author of Company Productivity, defines productivity as:

a family of ratios of quantity of output to quantity of input [Ref. 21: p. 2].

How is productivity defined for software development and maintenance? One expert defines it as follows:

> Programmer productivity is generally defined as the quantity of work produced by an individual programmer in a unit of time. . The definition implies the speed of programming, including the related tasks such as program design, coding, testing and documentation. The definition can be modified to use an expense or cost unit instead of a work unit. In addition, the definition should be extended to include program quality measurements. [Ref. 24: p. 18]

The following section will discuss programmer productivity in some detail with the emphasis on measurement aspects.

B. MEASUREMENT FOR IMPROVEMENT

1. Why We Should Measure Productivity

Capers Jones, author of several articles on programmer productivity and measurement, noted that preceding all the great advances in science in the 19th and 20th century were earlier advances in measurement instruments and measurement techniques. [Ref. 25: p. 39] Many authors speak of the evolution of software development from an art to a science. But are we there yet? Lord Kelvin's often quoted passage is appropriate here.

> When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you can not measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfying kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science. [Ref. 26: p. 16]

Perhaps we are not there yet but we are making progress. One thing is certain - unless we have systematic

methods to measure productivity, it will remain simply a buzzword with little or no real meaning. ISEC has done little to measure productivity as an organization. This makes it extremely difficult to gauge progress towards ISEC's goal "Foster Productivity."

We measure productivity in the software development and maintenance process primarily for management purposes. Productivity measurement provides information for planning, controlling and evaluating the entire process as well as the individual projects within the process. For planning, it provides management with information useful to estimate resource requirements. For control, planned resource estimates can be compared against actual expenditures. Variances can be analyzed and appropriate action can be taken to correct deficiencies. For evaluation, measurement systems provide valuable feedback on individual works, projects and the organization as a whole. [Ref. 27: pp. 33-35]

Most authors agree there are many potential benefits which accrue by measuring productivity. The benefits reflect a positive view of the purposes of productivity measurement. Table III summarizes the major benefits.

Lowell Arthur, author of Programmer Productivity, is a strong believer in measuring productivity for software development. Arthur contends that:

> One of the keys to improving productivity and quality is the ability to measure them. Software metrics provide a yardstick of system quality and project productivity. Without quantitative and qualitative measurement, you can't tell if your developing system is a lemon or a well-oiled machine. Furthermore, when it is operational you won't be able to tell what makes it a lemon or such an engineering marvel. You won't know where or what to fix or how to recreate the excellence of a previous system. You'll be no better off than blind men trying to describe an elephant. [Ref. 28: p. 125]

TABLE III

POTENTIAL BENEFITS OF MEASURING PRODUCTIVITY

ORGANIZATION

- It helps determine if the organization is getting its money's worth.
- It can improve the internal company climate.

MANAGEMENT

- Data collected can help estimate programming resources for scheduling a project.
- It facilitates management control.
- It can help indicate potentially high-cost, maintenance prone programs during the development.
- It can serve as a mechanism to help appraise and possibly reward employees.

PROGRAMMER

- It provides feedback on performance to employees.
- It can serve to motivate employees to higher performance levels.

2. Units of Measurement

a. Overview

There are two basic units of measurement, work units and cost units. Work units measure things like speed of programming; an example of which is lines of code (LOC) per man-month. Work units can be deceiving. They must be carefully defined and used cautiously to ensure comparability. An example of a cost unit is programmer-days per 1000 LOC. Because of inflation, cost units are not always stable. It is often advisable to use standard

dollars fixed to some base year instead of actual dollars to facilitate comparison when using cost units. [Ref. 24: pp. 18-19]

There are three basic types of productivity measures: partial factor productivity, total factor productivity, and total productivity. Partial factor productivity is the ratio of output to one class of input. Total factor productivity is the ratio of net output (total output minus intermediate goods and services purchased) to the sum of the associated labor and capital (factor) inputs. Total productivity is the ratio of total output (not net output) to the sum of all input factors. [Ref. 22: p. 7]

Management should be aware of some of the potential advantages and limitations of each type of measurement. Table IV and Table V provides a short summary of the advantages and limitations, respectively, to help in this regard. [Ref. 22: pp. 7-10]

b. General Problems with Measuring Productivity

There are several problems involved in the use of productivity measures. These center around the measurement of inputs and outputs and their abilities to measure efficiency. What is measured depends on the purpose (e.g. management intent) and use of the measurement. If management is interested in efficiency, when measuring inputs for use in a productivity measure, it is desirable to ensure that only the inputs that are actually utilized in the production process are used in the measure. This is especially important for the labor inputs. It implies, for example, that only time worked should be utilized in the measure instead of time paid. Although time paid is of interest for total cost purposes, elements such as administrative time should be separated out. Most

TABLE IV

ADVANTAGES OF THE 3 BASIC TYPES OF PRODUCTIVITY MEASURES

PARTIAL PRODUCTIVITY MEASURES

- They are easy to understand.
- Data is easy to obtain.
- Productivity indices are easy to compute.
- They are easy to sell to management because of the above 3 advantages.
- Some partial productivity indicator data is available industry-wide.
- They are good diagnostic tools to pinpoint areas for productivity improvement, if used along with productivity measures.

TOTAL FACTOR PRODUCTIVITY MEASURES

- The data from company records are relatively easy to obtain.
- They are usually appealing from a corporate economist's view.

TOTAL PRODUCTIVITY MEASURES

- A more accurate representation of the real economic picture of an organization is obtainable because they consider all the quantifyable output and input factors.
- If used with partial measures, they can direct management attention in an effective manner.
- Sensitivity analysis is easier to perform.
- They can be easily related to total costs.

organizations will probably want to track both time worked and time paid, however, as a pure efficiency measure, time worked is preferable. This separation will allow management to focus their efforts more appropriately. Care must be

TABLE V

LIMITATIONS OF THE 3 BASIC TYPES OF PRODUCTIVITY MEASURES

PARTIAL PRODUCTIVITY MEASURES

- If used alone, they can be very misleading and may lead to.costly mistakes.
- They do not have the ability to explain overall cost increases.
- They tend to shift the blame to the wrong areas of management control.

TOTAL FACTOR PRODUCTIVITY MEASURES

- When material costs form a sizable portion of total product costs, they are not appropriate.
- Only labor and capital inputs are considered.
- Data for comparison purposes is relatively difficult to obtain.

TOTAL PRODUCTIVITY MEASURES

- Data for computations are relatively difficult to obtain at the product and customer levels, unless data collection systems are designed for this purpose.
- As with the partial and total factor measures, it does not consider the intangible factors of output and input in a direct sense.

taken in the aggregation of inputs. Using labor as an example, different skills such as key punch operator versus a systems analyst, perform entirely different tasks. As such, their inputs should only be aggregated when they occur within a particular department. Where possible, it is preferable to measure inputs in terms of physical units rather than value units. [Ref. 27: pp. 37-38]

Exactly what should be measured can, indeed, be a problem. Convenient output measures are not always available in public sector organizations that provide a service where no acceptable definition of their outputs exist (e.g. national defense). In these organizations, most of the output measures in use are actually inputs to further processes - many are weak at best. In cases where inputs and outputs cannot be precisely measured, productivity measures become susceptible to manipulation and gaming. This implies that the control and evaluation phases of management may focus on faulty indicators. [Ref. 27: pp. 38-39]

Another related problem exists concerning how to deal with the quality of inputs. Idealy, to equitably compare various output levels, quality should be held constant. In reality, quality is rarely constant. In addition, quality changes are often difficult to measure. [Ref. 27: p. 39]

Dr. Barry Boehm cites a Weinberg study which found that programmers will tend to maximize (or minimize) whatever is being measured. Five different programming teams were given the same assignment but were given different directions about what to optimize while doing the job (e.g. minimize the number of statements or minimize the amount of memory required by the program). Four of the five teams finished first with respect to the objective they were asked to modify; the other team finished second. None of the teams performed consistently well on all objectives. The conclusion to be drawn from Weinberg's experiment is that management must carefully define the objectives for their programmers taking into consideration the conflicting nature of goals - programmers will try to otimize whatever is being measured. [Ref. 29: pp. 20-21]

A related problem is that programmers have been known to "pad" their output in effort to look better. There are many ways to do this. If lines of code per man-month is a critical management measure, programmers will tend to write programs that are longer than necessary without regard to machine efficiency. Programmers may even include duplicate loops or use other methods soley to increase their lines of code. Thus, we must always be aware that productivity measures are often susceptible to gaming. This is another reason why management must be very careful chosing: (1) what they measure; (2) why they measure it; (3) how they measure it; and especially (4) what management does with the results of the measurement.

In addition to the anomalies above, a few final words of caution and guidelines about productivity measurement are condensed below from various articles. The articles warn that:

1. Taking a measurement changes the system being measured (The Hawthorne Effect);
2. Comparison of results may be meaningless;
3. Results are sometimes paradoxical and misleading;
4. No single measurement tells the whole story;
5. Each measurement has its pros and cons;
6. Measuring productivity is not free; and
7. All measures are relative.

c. Measuring Programmer Productivity and Quality

(1) Preface. Keeping in mind the above discussion of productivity measures in general, we can now begin to discuss the measurement of programmer/project productivity. The difficulties of measurement are best verbalized by people who have actually struggled with the issue. Trevor Crossman complains:

> Programmer productivity is a dilemma. On the one hand, we want to control projects by knowing exactly when and where all slippages occur, we want to know if using a new methodology really is beneficial. We despise estimates that are based on "gut feel," but it appears that if we are to measure the productivity of our programmers we have to identify project variables and calculate their influence on our programming staff, make subjective assessments of the envisaged complexity of programs and the predicted ability of programmers, clarify terminology that has no industry-accepted definitions, measure the quality of our programmers' work, and base programmers performances on project estimates (which are arrived at unscientifically, anyway).
>
> It may be easier to say it just cannot be done. [Ref. 30: p. 144]

Crossman's comments demonstrate some of management's frustrations concerning productivity measurement. Some progress has been made though; many metrics have been proposed, tested, and found useful. These include measuring: (1) lines of code (LOC) per some labor unit; (2) functions which the user performs when utilizing the program; (3) functions which the program performs; (4) completed projects; and (5) quality and complexity. We will explore these major methods in the rest of this section. Clearly, many variations of these techniques exist.

(2) Lines of Code. Traditionally, software development organizations measured LOC per some unit of labor such as man-days or man-months. Many authors have suggested several problems with this general approach, however. Table VI is a summary of the shortcomings of LOC as a productivity measure.

Based on the strong objections to using LOC per labor unit as a productivity measure, one might conclude that it is useless. This is not so. Many companies, such as IBM, still use it as a management tool to gauge productivity despite its inherent shortcomings. Why? One major reason is that it is easy to measure. In fact, in many cases, the process can be automated. But because some

TABLE VI

PROBLEMS WITH LINES OF CODE AS A PRODUCTIVITY MEASURE

- There is no standard definition of LOC.
- LOC measurement is subject to gaming.
- LOC focuses attention on coding which is only 10-20% of the total software process.
- Comparisons across programming languages and companies are meaningless.
- Higher level languages are penalized.
- LOC does not address quality.

measure is easy to obtain is no reason, in and of itself, to use the measure. The critical question is "what is management's intent for use of the measure?"

Arthur recommends measuring executable lines of code (ELOC) to avoid the problem of lack of standardized definition of LOC. For organizations such as ISEC that program almost exclusively in COBOL, measuring ELOC amounts to counting only COBOL's verbs - statements that do something. Arthur provides a program in Appendix C of his book Programmer Productivity which automates the measurement process. Arthur contends that:

> ELOC provides the only valid measure of coding productivity currently available [Ref. 28: p. 133].

He freely admits, however, that ELOC suffers many of the same limitations as LOC. Yet management awareness and prudent judgement can overcome most of the limitations. [Ref. 28: pp. 132-135]

(3) Program Functions. Crossman, while working at the Standard Bank of South Africa, proposed and tested a productivity measure based on the number of functions within a program. He divided the number of man-hours spent during system development[3] by the sum of all programs in the system. [Ref. 30: pp. 144-5] Note that this really represents an inverse productivity measure. As in the case of LOC, program functions measure an input into the software development process instead of an output.

Precisely defining exactly what a function is may be a problem partly because Crossman's research involved only highly structured programs. In his article, "Taking The Measure of Programmer Productivity", he defines functions as:

> that section of the program that performs only one activity, such as initializing fields, computing values, setting up a print line, validating a record, etc.; has only one entry point and one exit point; conforms to the permited logic structure of structured programs; and has about 5-50 source statements [Ref. 30: p. 145].

The only factor that significantly affected the time to develop an application was the number of functions within a program. Crossman determined that he could disregard all other project variables for estimating the development time except for the use of "breakthrough technology" (e.g. new data base technology or a new operating system). In those cases where new technology was employed, the development time doubled indicating a steep learning curve adapting to new technology. The management implication is that program functions may be a useful planning and resource estimating tool. On the other hand,

---

[3]Crossman defines development time as design, code, inspection, and unit test but excludes system test which he feels rarely is a development time/cost driver.

program functions suffer many of the same limitations as other methods. [Ref. 30: pp. 145-147]

(4) External Attribute Functions. Allan J. Albrect, working at IBM's DP Services Organization, pioneered another approach to the programmer productivity measurement dilemma. Albrect proposed a measure based upon the external attributes or functions that a software product involved. The general approach is to count the following features in an application: (1) user inputs; (2) external inquiries; (3) external outputs; (4) master files; and (5) system-to-system interfaces. [Ref. 31: p. 102]

The subtotals of the individual five factors are weighted (by trial and error) by numbers designed to reflect the function value to the user. The weighted totals are then added. Additional adjustments can be made to account for a particular project's quirks. For example, if an application is particularly complex, the total can be increased by some percentage. The result is a dimensionless number defined in "function points" which Albrect has found to be an effective relative measure of function value. The actual measure used is hours worked per function count, another inverse productivity measure.

Albrect asserts that function value is programming language and technology independent. The measure of external attribute functions offers the considerable advantage of actually attempting to measure the results of the entire software process. In this respect, it corresponds more closely to a true productivity measure. It is also less subject to gaming than other methods. One author points out:

> What is significant is that function points do not contradict what would have been speculated, lending credibility to this concept of measurement. Without this credibility, future productivity assessments would not be possible. [Ref. 31: p. 108]

Another author counters with:

> The disadvantage of function points is that they are imprecise and often misunderstood. Many people perceive a function point figure to represent function delivered to the user. In fact, it represents the amount of function imbedded in the specific design of the system; much of the imbedded function may be invisible or not utilized by the user. Indeed, different designs meeting the same requirements may have widely different function point counts. [Ref. 32: pp. 134-135]

(5) Completed Projects. Another possible measure is completed projects per unit of labor. The definition of project would need clarification but the method does appear easy to implement and use. To make it a viable measure, managers would have to ensure that employees were given projects of equal difficulty over some period of time. If employees were left to select their own projects, typically only the short, easy or interesting projects would get done. Difficult projects with potential high payoff to ISEC may lay dormant at the bottom of some in-box. Some type of weighting scheme could be used based on management's judgement as to the difficulty or importance of the project. The requirement to balance the load equitably among the employees is no easy management task and may offset the ease of implementation and ease of use advantages. [Ref. 27: p. 47]

(6) Complexity and Quality Metrics. The work of Maurice H. Halstead (1977) and Thomas McCabe (1976) has given birth to a another view of productivity measurements.[4] Halstead developed a number of metrics that are computed from easily obtained properties of the source code (e.g. the total number of operations in a program). The metrics he

---

[4]A complete description of these metrics is beyond the scope of this thesis. They can be found in most modern text books on software design or software engineering including Richard Fairley's Software Engineering Concepts, McGraw-Hill, Inc., 1985.

proposed were a program length metric, a program volume metric, and a program effort metric. Followup research has proven that Halstead's effort metric is well correlated with the observed effort required to debug and modify small programs. Program effort thus appears to be a measure of interest for software maintenance. [Ref. 33: pp. 323-324]

McCabe observed that the difficulty of understanding a program is strongly influenced by the control flow for that program. McCabe's metric is based on graph theory but really amounts to adding the number of logical operators (AND, OR, and NOT) to the number of decisions. To keep errors to a minimum, he recommends an upper bound of 10 as the maximum complexity for the control graph of an individual routine. McCabe's original research demonstrated strong correlation between cyclomatic complexity, ease of testing and the reliability of the routine. Thus, McCabe's metric can help identify those modules that are candidates for rewrite. [Ref. 33: pp. 324-325]

Arthur offers two complexity measures for COBOL programs. The first metric he suggests is to sum all of the CALL, PERFORM, SORT, MERGE, COMPUTE, INSPECT, and GENERATE statements and divide that result by LOC/100. Dividing by the 100 normalizes the metric for ease of comparison with other programs. (There is nothing magic about the number 100, it could just as well be 50.) This, he claims, provides a metric called 'function density' which is actually a complexity measure. The higher the functional density, the more functional and modular the program. [Ref. 28: pp. 135-6]

Arthur's second complexity measure counts the number of decisions in the program. The sum of the IF, PERFORM UNTIL, PERFORM TIMES, and SEARCH WHEN counts gives the total number of decisions in the module. This

represents a basic measure of program complexity and testability. This sum is then divided by the total LOC in the program, and again is normalized by dividing by 100. This metric is known as "decision density"; it provides the number of decisions in each 100 LOC. Arthur claims decision density is a highly representative measure of complexity, "the cruel task master of maintenance programming."

In addition to measures that focus primarily on the coding phase, there are other quality metrics for documentation, testing, etc. Productivity measures for documentation include: (1) cost per documentation page; (2) document pages per unit of time; and (3) document cost per 1000 LOC. Productivity measures for testing might include: (1) test cases developed and executed per unit of time; and (2) cost per defect. Note that these are partial factor productivity measures. As such, they probably can and should be used but only with extreme caution - they are all susceptible to manipulation and gaming. They may influence employee behavior, depending on management and the incentives program established, to maximize some ratio at the detriment to the total process and the organization.

It is appropriate to end this section with a quote from Tom Gilb, author of Software Metrics. Gilb says:

> Again, we are forced to recognize that, although many readers might be tempted to argue "I can't go around and measure everything. My programmers have too much work to do already," the introduction of appropriate measuring techniques does not cost, it saves. It is not a luxury, it is a necessity. [Ref. 34: p. 64]

It is amusing to note that Gilb dedicated his book to all the people who have patiently explained to him why it is "impossible", "impractical", or "uneconomical" to measure software quality!

C. IMPLEMENTATION OF A PRODUCTIVITY MEASUREMENT SYSTEM

1. Overview

The previous sections have discussed the "why" and the "what" of productivity measurement; we now proceed to the "how" - that is, implementing a system to monitor performance. Why doesn't every organization have a measurement system if there are so many benefits? As previously mentioned, the collection and analysis of this information is not free. It requires machine, people, time, and other resources. Automation of the process may help to lower administrative costs but it doesn't eliminate them. Another problem is people's natural resistance to change. In addition, some managers and employees may feel threatened working in an environment where their actions are recorded and documented. These human issues must be given appropriate attention. But as Gilb pointed out above, tracking productivity information is cost-effective in the long-run - it is a necessity!

Capturing this information may be helpful for reasons other than gauging performance. A measurement system cam provide quantitative justification of resources required for particular projects. Under the commercial activities program, all non-mission-essential activities are subject to private sector provision. For software development and maintenance, this program would require ISEC to bid on particular projects along with commercial software houses. Such bids must be auditable, which implies that productivity information must be quantitatively-based and verifiable. [Ref. 27: p. 49]

2. Implementing the Measurement System

The following subsection discusses the implementation process for an actual measurement system at

ISEC. It has been adapted from and is based on the ideas of Irving Siegel, author of Company Policy. [Ref. 21: pp. 45-53] Implementation of a measurement system consists of the following 6 basic steps:

1. Top Management Commitment;
2. Task Force Selection and Charter;
3. Marketing the Program;
4. Collection of Information;
5. Designing the System; and
6. System Installation and Evolution.

Each phase is discussed below.

a. Top Management Commitment

As with most systems, unless the top brass is commited to it, the chances for success are dismal. To obtain top level commitment, it may be prudent to establish a pilot program in one of the programming directorates. The idea should be to collect data on a number of software projects and evaluate several measures using this data. Guidance should be solicited during this initial stage concerning any restrictions or constraints which top management may feel are appropriate.

b. Task Force Selection and Charter

The second step is the selection of a task force or steering committee and developing the organizational charter. Members should be strategically chosen. They should represent a broad-class of organizational skills required for software development and maintenance. Some top level participation is advisable. The charter should be formulated with an overall objective of devising a monitoring scheme that satisfies company needs and meets the stated time and cost constraints.

c. Marketing the Program

The third step is carefully marketing the intent of the program to all levels at ISEC. The fears and anxieties of managers and the rank-and-file employees need to be quelled before the rumor mill begins to churn. Briefings, seminars, fact sheets, and the ISEC newspaper should discuss the program and its purpose completely and candidly. ISEC should consider designating productivity officers at various levels to serve as liaison up and down the chain.

d. Collection of Information

This fourth step may be more accurately described as "doing your homework." ISEC's data bases and skill resources should be studied. The lessons learned and suggestions for improvement from the pilot program should be documented for later use in designing the actual measurement system. The programs of other government agencies should be ascertained and evaluated. In particular, the General Accounting Office (GAO), the General Services Administration (GSA), the National Bureau of Standards (NBS), the National Security Agency (NSA), U.S Air Force and Navy should be surveyed to learn from their experiences. Neglecting these resources would be a serious blunder. Another possible source of information are the productivity offices that each of the military services has. A thorough and careful analysis of what information is currently being collected should be conducted. Things often overlooked such as "who will train management and the employees?" and "what will the training consist of?" need to be addressed during this phase. It is possible that ISEC does not have the in-house assets to accomplish these myriad tasks. It may be necessary to seek the assistance of a consultant.

e. Designing the System

The main objective of the task force is to arrive at a first-generation monitoring system that satisfies ISEC needs and constraints. This implies the system is not static but evolutionary. All the lessons learned from the pilot system should be considered to make the transition process as smooth and painless as possible. ISEC employee suggestions and the results of surveying other federal agencies should also be considered. Automation of the administrative accounting and record keeping system should be "designed in" to the maximum extent possible. A preliminary users manual should be written to guide the operators of the system. It should describe the nature of the system and its structure as well as covering the measurement process itself and the procedures for carrying it out.

f. System Installation and Evolution

It is probably wise to designate the first six months or so as a trial period to work out the bugs, refine the procedures and seek suggestions for improvement. The process is much like developing STAMMIS for users. It is an iterative process. Not everything can be prespecified in advance. The trial system will stimulate new and better ways to measure productivity.

D. SUMMARY

This chapter provided definitions for productivity. Productivity was distinguished from production, efficiency and effectiveness. The benefits for measuring productivity were explained and the problems with monitoring productivity were discussed. Some guidelines were included concerning the use of productivity measures. Specific productivity

measures for software development and maintenance were evaluated in light of their advantages and disadvantages. Finally, a strategy for implementing a productivity measurement system at ISEC was suggested based on the ideas of Irving Siegel.

The next chapter discusses the problems with traditional methodologies for obtaining accurate and complete specifications. Prototyping and evolutionary development are explained and recommended as an alternative to more conventional techniques.

IV. REQUIREMENTS ANALYSIS AND SYSTEMS DEVELOPMENT WITH PROTOTYPING

A. PROBLEMS WITH THE TRADITIONAL SOFTWARE LIFE CYCLE

1. The Traditional Software Life Cycle

Internal documents at ISEC reveal that the average development time for standard systems is five to seven years [Ref. 35]. This translates to users that are handcuffed by inefficient and ineffective systems. It means managers are not able to get the decision making information that they need. With the continual rapid advancements in hardware technology, it also means the system will be fielded on obsolete hardware.

What causes a software development time of five to seven years? Software development is a complex process so there are no simple answers. Reviewing the software life cycle may be helpful. Figure 4.1 is a representative version of the traditional software life cycle. [Ref. 36: p. 29] Figure 4.2 shows the phases of the DOD system life cycle. [Ref. 2: p. 302]
As these figures show,

> The software development cycle is often presented as a sequential set of well defined phases, each with specific products and reviews, which provide the necessary structure to facilitate management and control by the developer/project manager. [Ref. 37: pp. 74-5]

Despite this phased approach, the end result has frequently been software that is late, over budget, or does not meet user needs.

Many software professionals feel the life cycle itself is the root of the problem. Daniel McCracken and

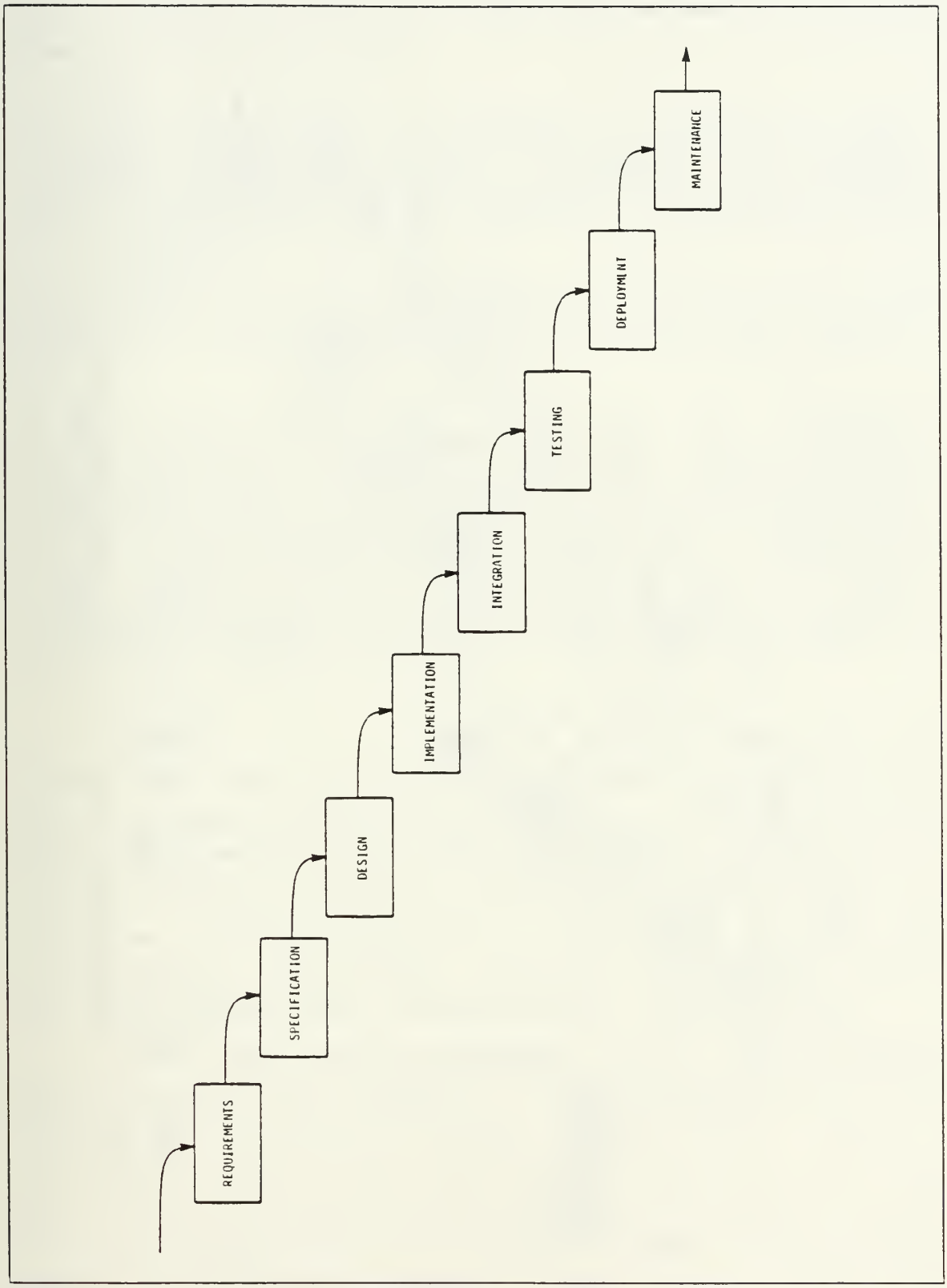

Figure 4.1 The Traditional Software Life Cycle.

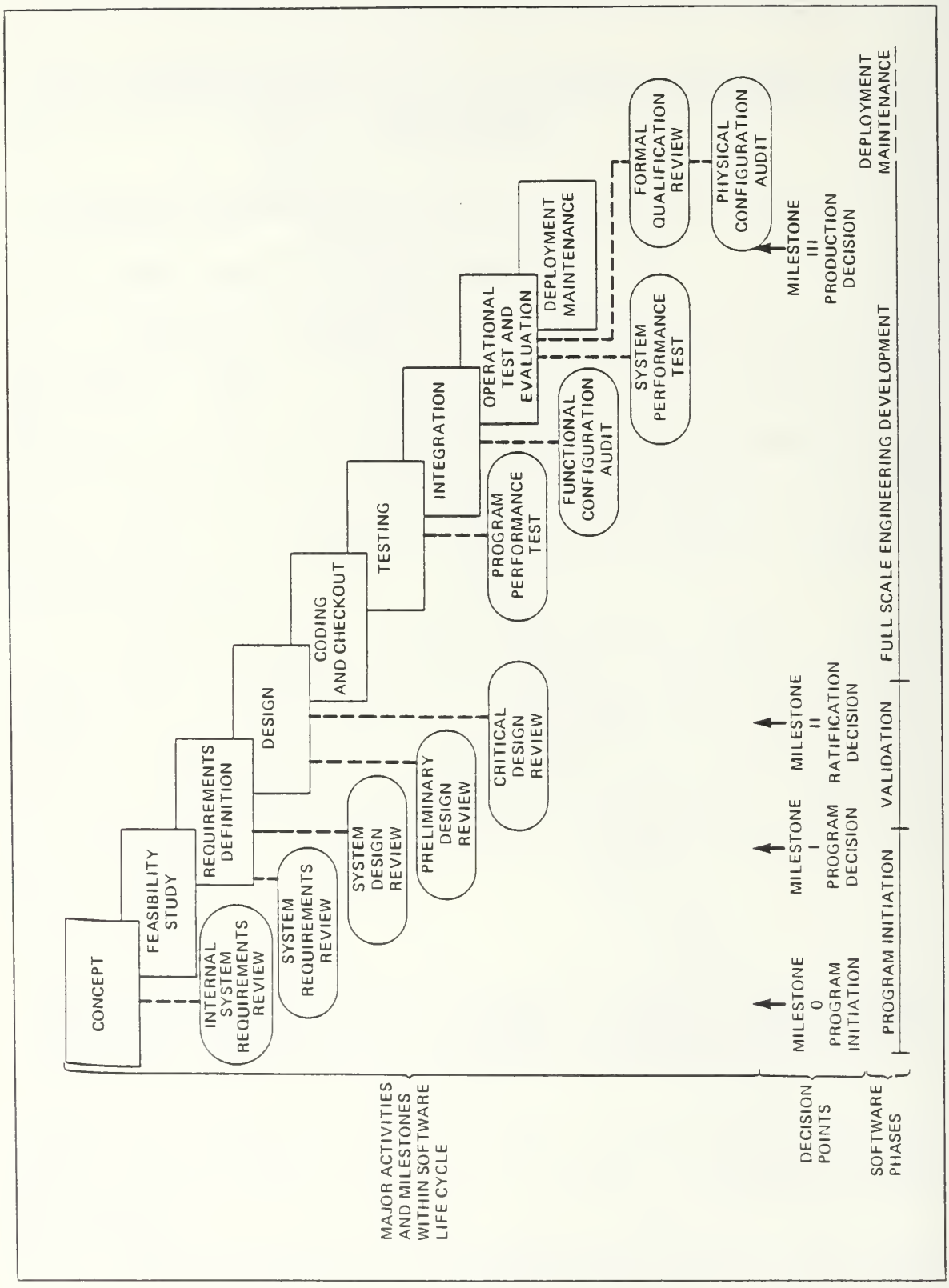

Figure 4.2 The Department of Defense System Life Cycle.

Michael Jackson, authors of "Life-Cycle Concept Considered Harmful", put it this way:

> The life cycle concept perpetuates our failures so far, as an industry, to build an effective bridge across the communications gap between end-users and systems analysts. In many ways it constrains future thinking to fit the mold created in response to failures of the past. [Ref. 38: p. 30]

.

Along these same lines, G. R. Gladden, Supervisor of Quality Assurance at Honeywell's Build Services Division, warns:

> I am of the opinion that the concept of a 'software life-cycle' is no longer helpful, indeed may be harmful to our software development profession. In its various forms the life cycle has sought to describe the software development process as iterative events within the major tasks of design, implementation, test, etc. One begins to visualize the development process as a sequence of tasks 'waterfalling' into one another while within each task modifications occur iteratively as a better understanding of the system is acquired. These interactions work together to extend project schedules, invalidate designs, alter test requirements, and to generally infuriate customers. [Ref. 39: p. 35]

ISEC, in an effort to diminish the "software crisis", has tried a variety of techniques and methodologies that have helped improve the situation. They attacked the problem by standardizing their efforts and formalizing the process. They attempted various structured approaches including structured analysis. Voluminous specification documents were "ping-ponged" back and forth between the FP and ISEC. Regrettably, major problems remained. Why?

The specification document is the foundation on which the software is built. It is fundamental to the conventional software life cycle. James Martin, author of the world's best selling computer books, believes that there are serious problems with specification documents and the process by which they are validated. Table VII lists the major problems of the process. [Ref. 2: p. 7] There are

many who share Martin's thoughts. McCracken and Jackson write:

> systems requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't know them in advance - not even in principle. To assert otherwise is to ignore the fact that the development process itself changes the users perceptions of what is possible, increases his or her insight into applications environment, and indeed often changes that environment itself. We suggest an analogy with the Heisenberg Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose. Systems development methodology must take into account that the user, and his or her needs and environment, change during the process. [Ref. 38: p. 31]

TABLE VII

PROBLEMS WITH THE SPECIFICATION PROCESS

- It lacks precision. It cannot be converted into computer code without many assumptions and interpretations.
- It contains many ambiguities and inconsistencies.
- It is usually incomplete.
- It is often so long and boring that key managers do not read it. They read only the summary.
- It is often misinterpreted by both sides. Often its readers think they understand it but in fact do not.
- Sometimes much trivia and motherhood are added to the document. Both sides understand this. It increases the comfort level, but has zero value.
- The specification document is not designed for successive refinement as the problems become better understood. It is intended to be a complete document which users sign.

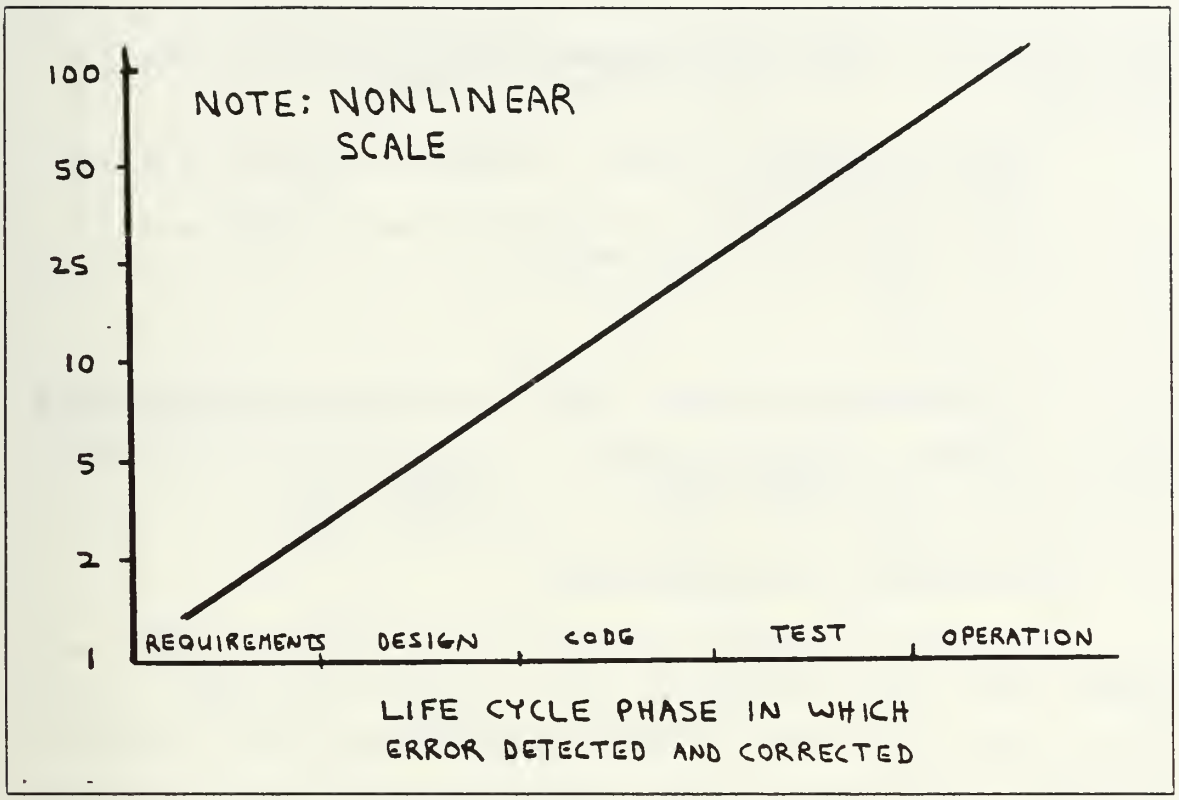

Figure 4.3 Relative Cost to Correct Errors in Software Life Cycle.

What do these specification problems mean in dollars? Figure 4.3 shows the relative cost of correcting a requirement or design error as the product progresses further into development. It helps illustrate the high level of risk involved with the traditional software life cycle. When errors are found late in the cycle, requirements have to be revalidated, design redone, software and system retested, and documentation rewritten. In his article "Seven Basic Principles of Software Engineering," Dr. Barry Boehm supports this idea when he writes:

> There is one single message about developing reliable software which out weighes all the others. It is to get the errors out early.
>
> One of the most prevalent and costly mistakes made on software projects today is to defer the activity of

detecting and correcting software problems until late in the project. There are two main reasons why this is a mistake:

1. Most of the errors have already been made before coding begins; and
2. The later an error is detected and corrected, the more expensive it becomes.

[Ref. 40: p. 9]

The cost to correct increases dramatically as we move from phase to phase. Errors traced to specification documents are very expensive to fix.

2. Maintenance Considerations

We need to look beyond just software development and consider the entire software life cycle. The goal is to minimize total life cycle costs. Minimizing only software development costs may cause suboptimization and possible higher total costs.

> It is well established that maintenance activities consume a large portion of the total life cycle budget. It is not uncommon for software maintenance to account for 70 percent of total life cycle costs (with development receiving 30 percent) [Ref. 33: p. 311].

Figure 4.4 is a graphic portrayal of these facts. Software maintenance[5] involves developing enhancements, adapting to new environments, and correcting problems.

The following quote from Software Engineering Concepts captures the essence of the activities in the maintenance phase:

> Software product enhancements may involve providing new functional capabilities, improving user displays and modes of interaction, upgrading external documents and internal documentation, or upgrading performance

[5]Some software engineers prefer the term evolution to maintenance. Although evolution may be a more accurate term for this phase, the more traditional term will be used here.

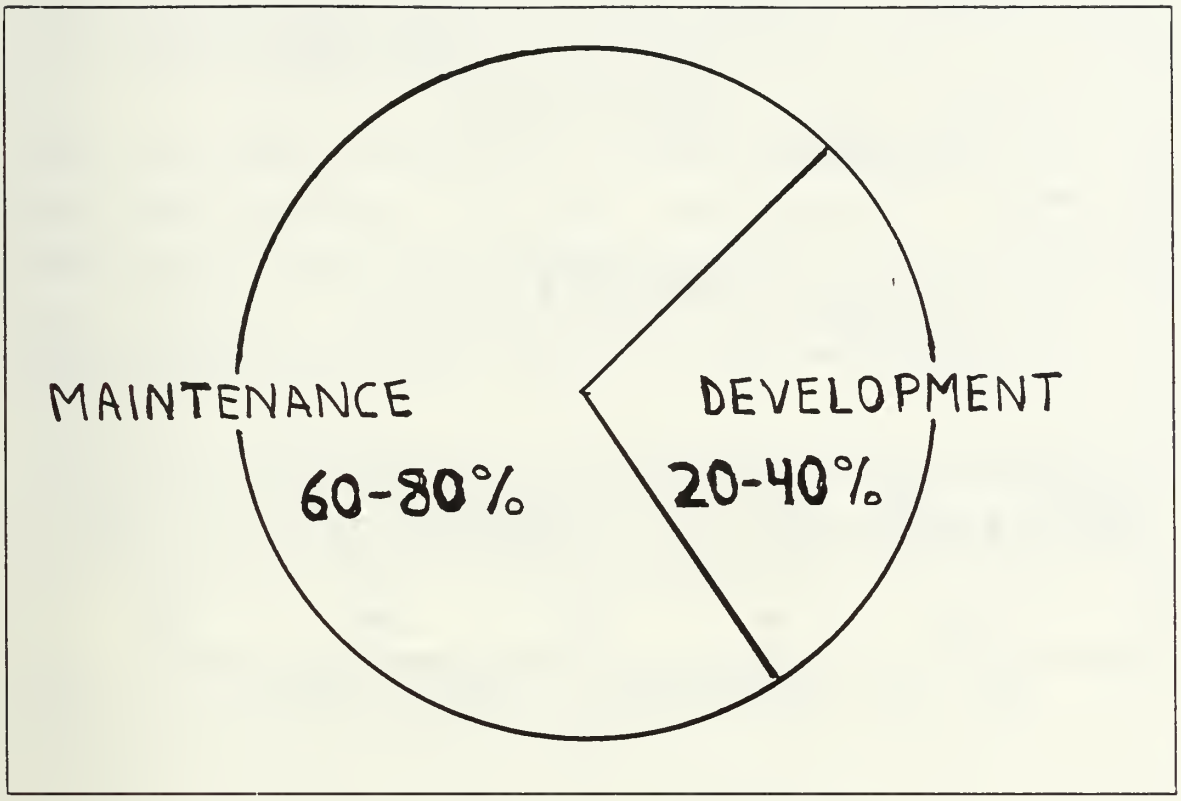

Figure 4.4 Maintenance - The Largest Cost Driver.

> characteristics of a system. Adaption of software to a new environment may involve moving the software to a different machine, or for instance, modifying the software to accommodate a new telecommunications protocol or an additional disk drive. Problem correction involves modification and revalidation of software to correct errors. [Ref. 33: p. 311]

It has been estimated that 60 percent of the maintenance budget involves enhancements while adaption and correction each account for 20 percent of the maintenance effort (See Figure 4.5). If the above statistics are correct, we can conclude that over 40 percent of all life cycle costs are for product enhancements. Why such a high percentage for enhancements? A major contributing factor is that formal, structured techniques have not provided software developers with complete or accurate design

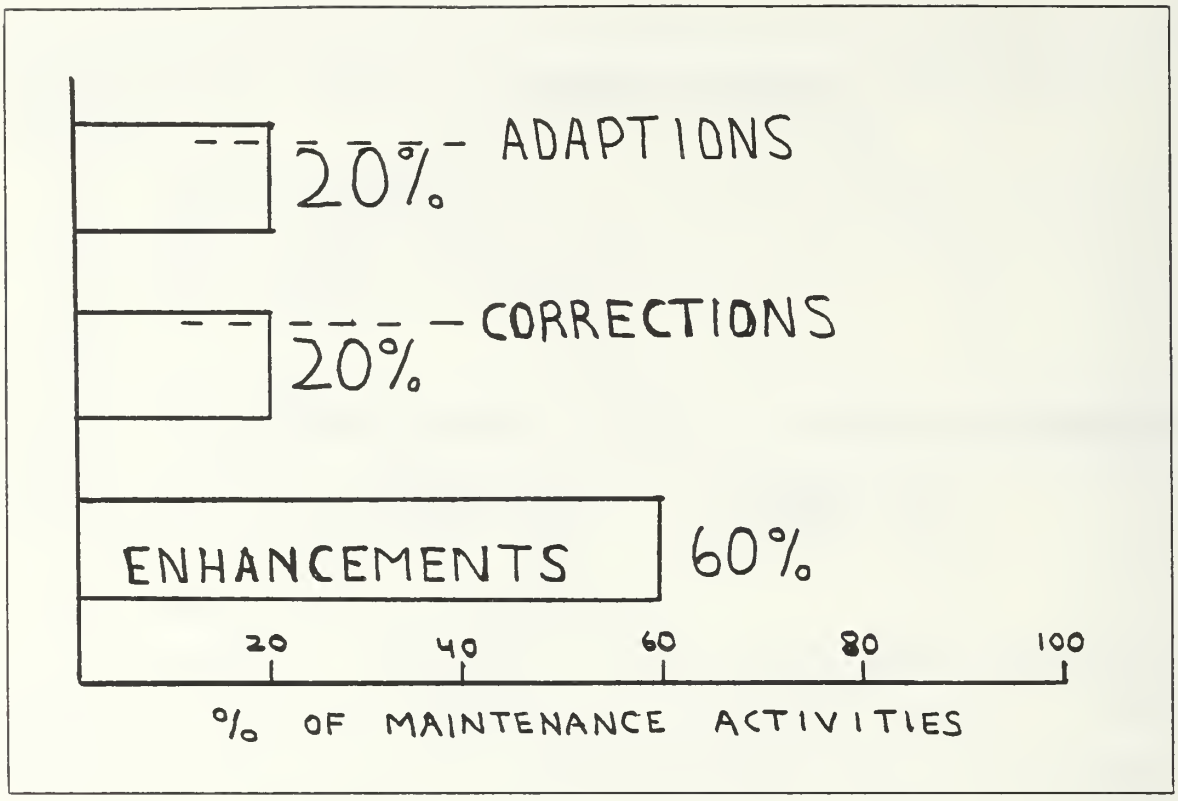

Figure 4.5 Relative Effort for Maintenance Activities.

specifications.[6] A methodology or tool that could provide accurate and complete specifications would reduce the effort necessary for product enhancement. This, in turn, would decrease maintenance costs thus reducing total life cycle costs.

To increase productivity, ISEC must change their traditional way of developing STAMMIS. This rather bold statement accepts "the challenge" of the Commander of USAISC, Lieutenant General Emmett Paige Jr., who declared:

---

[6]Few would argue that unstructured code is better than structured code. Structured techniques have created software that is easier to understand and maintain. Structured techniques have been less successful, however, providing clear, concise, consistent, and complete design requirements.

> We need to change the way we do business - challenge the 'way we've always done it' - and develop innovative ways to field systems sooner [Ref. 41].

ISEC must seek to bridge the communications gap between the functional proponent, the ASD, and the user. Traditional requirements analysis techniques have resulted in a document that few people read or understand. Bernard Boar, author of Application Prototyping, makes the following comment:

> If you are serious about alleviating the productivity problems with application development, there is only one question that deserves your attention: What technique offers the highest probability of delivering a clear, correct, consistent and validated requirement statement of the user's need? [Ref. 42: p. 29]

Boar's answer is, of course, prototyping. We will explore prototyping in subsequent sections in this chapter.

B. PROTOTYPING AND EVOLUTIONARY DEVELOPMENT

As the previous section has illustrated, the net result of improper requirements analysis are increased costs, duplicated efforts and a poor product. A definition of requirements and specifications is probably overdue. Requirements provide an understanding of the general applications area and should include a list of desirable features that the system should contain. Specifications record precisely what the function of the system is. It is derived from the requirements. Specifications normally do not involve "how to" implementation details. [Ref. 43: p. 4]

A prototype is nothing more than a model or pilot system. Prototyping is not a new concept. Scientists and engineers learned long ago that models and pilot systems are necessary and useful learning tools which lessen the inherent technical risks associated with developing new

systems. Software developers have been slow to use this approach when building application programs.

Fred Brooks, respected author of The Mythical Man-Month and project manager for the IBM 360 operating system, writes

> The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver a throwaway to the customers. Seen this way, the answer is much clearer. Delivering that throwaway to the customer buys time, but it does so only at the cost of agony for the user, distraction for the builder while they do the redesign, and a bad reputation for the product that the best redesign will find hard to live down.
>
> Hence plan to throw one away; you will anyhow. [Ref. 44: p. 116]

There are two basic views of prototyping. One is the Fred Brooks' view. That is, the prototype is seen as a requirement specification-, technical feasibility-, and/or requirements validation tool. It's a throwaway and nothing more. The other view of prototyping is known by several names; incremental development, iterative development, iterative refinement, and evolutionary development are the more common names. Dr. Boehm describes incremental development as:

> Development of a software product in several expanding increments of functional capability, as a means of hedging against development risks, of smoothing out the project's personnel requirements, and of getting something useful out early [Ref. 40: p. 8].

The throwaway view of prototyping is the older of the two views yet it represented a radical change to software development practices in the early and mid 1970s. Except for the most progressive software developers, very few accepted prototyping as a development methodology. With the advent of fourth generation or non-procedural languages in the late

1970s and early 1980s, the second view of prototyping became technically feasible. Yet as Boar wrote in 1984, "Prototyping of large systems would not have been possible just two years ago." Thus, as advancements have been made in fourth generation languages, the real value for applications development has only recently been established.

C. ADVANTAGES OF PROTOTYPING

The literature on prototyping suggests both qualitative and quantitative benefits of prototyping. Table VIII summarizes the major advantages suggested by authors who have hands on experience with prototyping.

Experience to date shows that prototyping is usually faster than traditional methods but not always. The main benefit of prototyping is the role it plays bridging the communications barriers in the development process. There exist cultural differences between the user, ASD, and the FP. These are the result of technical and organizational differences. The prototype helps to create a common framework from which to work. Specifications are better because they are in a form the user can realistically validate. It frees the user and the FP from being forced to sign off on reams of paper that they vaguely comprehend. [Ref. 45: pp. 38-46]

Because the user is an active participant, there are significant benefits resulting. The man-machine interfaces can be tested early in the life cycle and adjusted as necessary. As users gain experience with the prototype, they can provide valuable feedback for changes and enhancements. Users can change their mind; specifications are not locked in concrete so early in the process that good ideas are frozen out. There is a natural bonding that occurs during the development, feedback, and testing

TABLE VIII

ADVANTAGES OF PROTOTYPING

- Provides a facility to permit assessment of the impact of the system on the whole user environment.
- Forces a user centered approach. Users can change their mind. User acceptance is easier to obtain.
- Permits early testing of human/machine interfaces.
- Helps alleviate project communication problems caused by cultural differences.
- Provides a medium for validating requirements.
- Requires fewer programmers.
- Potentially decreases development time.
- Reduces technical risks.
- Reduces maintenance costs thus reducing life cycle costs.
- Stimulates programmers and increases their motivation level.

process. Users perceive (quite correctly) that they are an integral part of the system. In addition, users develop a "warm feeling" for the end product, hence, their confidence increases which facilitates acceptance of the system. [Ref. 46: pp. 15-18]

This author believes that the greatest potential benefits of prototyping will be for large systems development. These are the projects with the longest development lead times and have the highest degree of risk associated with them. An incremental approach where the heart of the system is developed first and then embellishments and refinements are added should reduce

development risk and total life cycle costs. Designs and ideas that do not work out can be scrapped at minimal cost and embarrassment to the developers.

D. PROTOTYPING LIMITATIONS

Despite the numerous advantages of prototyping, it does have limitations. Prototyping may frustrate users that have seen a working model. Some users may want to implement the prototype directly - something it is not normally designed to do. Experience with prototyping at the Del Monte Corporation resulted in the following observation:

> Systems development learned that they (the user) thought the application was 90% complete. But this was not so - the prototype only simulated the operation of the system. Users had little concept of the amount of work needed to complete a prototype - adding validation and editing routines, implementing the database design, adding security, backup, and recovery features, turning it over to operations and maintenance programmers, and so on. Users often become impatient when development of these 'back end' portions appeared to be taking too long. [Ref. 47: p. 2]

Del Monte used two approaches to counteract this user reaction. First, they "phase implement" the system creating the critical portions initially and adding functions incrementally. Second, they have end users assist in the programming. They also try and reduce the system to the smallest version that will still meet basic user requirements but may lack the "bells and whistles." [Ref. 47: pp. 2-3]

Another limitation of incremental development is that prototypes are not developed with efficiency in mind. The idea is to minimize human resources by developing the software faster and using less programmers. It is possible to build hybrid systems today which take advantage of the efficiency of COBOL and the coding speed of fourth generation languages.

TABLE IX

LIMITATIONS OF PROTOTYPING

- Prototyping does not necessarily result in shorter development time.
- Because of the iteration process, resource planning and time estimating procedures are difficult.
- There exists a possibility that something major may be left out of the system.
- The issue of when to stop the iteration process is not clear.
- Some users may get bored or irritated if the iteration process takes too long or if the initial prototype is way off the mark.
- Some users may want to implement the prototype directly.
- Prototyping may require a substantial investment for the procurement of software tools, training costs and additional computer hardware.
- Prototyping does not always use computer resources efficiently.
- Prototyping is not appropriate for all types of applications.
- Programmers may feel threatened by prototyping.

Prototyping does not allow the software developer to throw away software engineering principles. There is still a need to do a preliminary requirements analysis prior to building the prototype. The larger system environmental constraints will eventually have to be reckoned with. Developers who ignore documentation during evolutionary development will discover in the maintenance phase that their "shortcuts" may actually increase total system costs! Rigorous structured techniques are still important to ensure all of the bases have been covered.

Other authors have suggested additional limitations of prototyping. Table IX is a summary of prototyping limitations.

Whether all the above complaints, issues and limitations are valid is a matter of debate. Regardless, prototyping is not a panacea. Its limitations must be considered before applying it to a particular problem.

E. A BRIEF OVERVIEW OF THE PROTOTYPING LIFE CYCLE

Figure 4.6 shows a proposed software development life cycle which acknowledges the effect of prototyping on the development process [Ref. 45: p. 105]. The Feasibility Phase is the same as in the traditional life cycle model. During the feasibility phase, typically a preliminary cost/benefit analysis is done as well as determining the applicability of a computer-based solution. If the application is appropriate, the Prototype Phase seeks to develop user requirements.

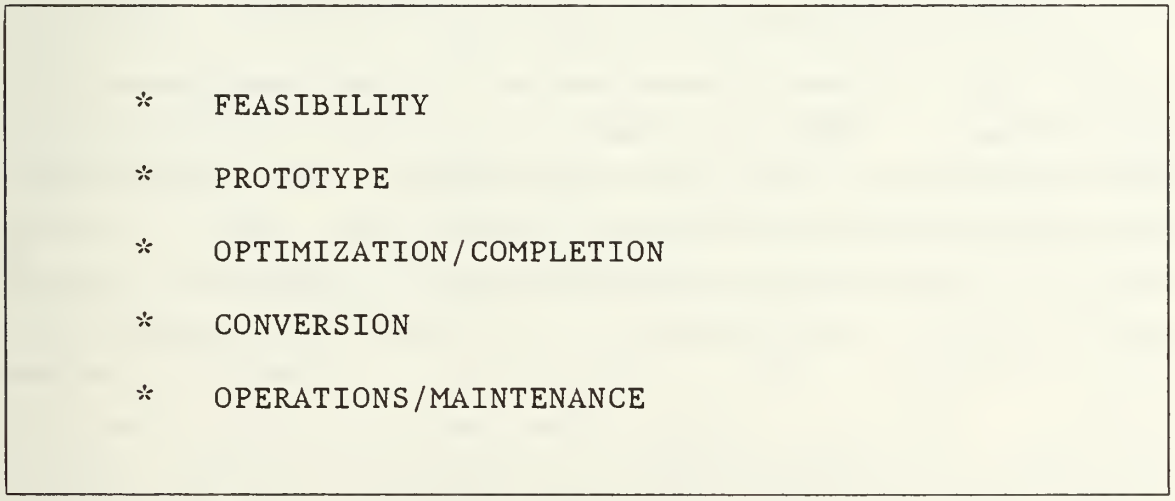

Figure 4.6 The Prototyping Life Cycle.

Prototypes initially ignore many larger, system constraints. The Optimization/Completion Phase brings the prototype into harmony with any operational constraints. It is usually neither feasible nor prudent to implement a prototype directly. Software that is appropriate for prototyping may not be appropriate for production architectures. Efficiency, data base size, and transaction processing rates are considerations for the actual systems which are largely ignored by the prototype. The Conversion Phase addresses these types of issues. [Ref. 45: p. 104]

The final phase, Operations/Maintenance is similar to its counterpart in the conventional life cycle. Iteration does not magically stop when the system is fielded. Laws and regulatory changes, user needs, and environmental changes will cause the STAMMIS to evolve. It is important to consider what changes in the software are likely to be made when the prototype is built. The software should be written to accommodate such change and thus reduce maintenance costs. [Ref. 48: pp. 226-235]

F. PROTOTYPING APPLICABILITY AND IMPLEMENTATION AT ISEC

Not all system structures are good candidates for prototyping. Applications that are extensively batch-oriented are probably inappropriate for prototyping. Algorithm-based problems and problems with limited transaction processing but which require considerable number crunching power do not create a conducive environment for rapid iteration. Many of the Army's STAMMIS are batch systems. This is probably due more to prior hardware constraints and when the MIS were developed rather than a blanket statement that the Army prefers batch MIS to on-line systems. User and management needs would be better served in most cases if they were on-line or real time systems.

A big question, then, is whether prototyping is appropriate for STAMMIS development. Management information systems tend to deal with structured problems. Based on his experience at American Telephone and Telegraph (AT&T), Boar sums up prototyping candidates this way:

> Prototyping works best for on-line transaction processing oriented applications. The application should be a structured problem with a large amount of data elements and record relationships but a small amount of algorithm processes. [Ref. 45: p. 64]

Nearly all STAMMIS meet these criteria.

Given the two views of prototyping discussed earlier, which view of prototyping is appropriate for ISEC? Perhaps both are appropriate. The throwaway prototype is applicable to STAMMIS projects that, for whatever reason, must be contracted out. The skeleton system developed during the prototyping process can be used as a requirements document. It may also be an appropriate view for a STAMMIS which requires a degree of efficiency that current fourth generation languages do not provide. In both cases, the prototype serves as a requirements/specification document.

The evolutionary or iterative approach is appropriate for most STAMMIS developed in-house at ISEC, regardless of application size. Some literature on prototyping suggests that it will only support small systems development. [Ref. 49: p. ID/9] This may have been true in 1978 but there is evidence that this is no longer true today. Boar contends that:

> Given the state of software technology today, there is no reason why the techniques described in this book cannot be used to build rapid prototypes of medium and large size applications as well as simple ones [Ref. 45: p. 12].

ISEC has done some recent experimenting with prototyping. The ASD assigned to support the Logistics Center at Ft Lee, Virginia, has successfully built two logistics systems using an incremental development strategy.[7] Their successes are a testimony that prototyping works. The initial test system, the Standard Army Retail Supply System (SARSS), was created using a technique known by the acronym STEP-UP (Systems Through an Evolutionary Process Using Prototyping). SARSS was no trivial project. Yet in only four months, they had designed, developed, and tested a system of nearly 175,000 lines of code. SARSS was then demonstrated to users and, through user feedback, on-the-spot improvements and enhancements were made. ISEC estimated that the system will be fielded in two years (from project initiation) compared with the average track record of 5 to 7 years.

SARSS was a bold step for ISEC. To attempt such a venture meant largely ignoring normal Army and DOD system development policies. The other success story is the Unit Level Logistics System (ULLS). Two more systems are being prototyped and are scheduled for fielding within the next year. All of these systems are interactive systems employing the latest microcomputer technology. They can be run from menus or with a command language thus supporting both beginner and experienced users. They also feature extensive "help" facilities. [Ref. 50: pp. 1-14]

Why were these systems successful? There are several reasons; they are listed below.

---

[7]The ISEC ASD that supports logistics systems is physically collocated with the FP in the same building. This is not true of the ASDs that support personnel and financial systems. This author believes that it was a fundamental reason why the two projects succeeded.

1. The physical collocation of the FP and ASD helped foster a team attitude and facilitated problem solving, and communication problems. The "We - They Syndrome" was eliminated.
2. Good management and software engineering techniques were employed. Careful planning and quality assurance activities such as walkthroughs and testing at multiple levels, a "murder board" and change control techniques were conducted.
3. The users were an integral part of the process. User feedback was sought at various phases and their contributions in terms of improvements and enhancements were significant.
4. Prototyping represented a new challenge to the ASD supporting logistical systems. It was exciting to work with an innovative software development methodology. This challenge and excitement led to employee enthusiasm and internally-driven motivation.

There are some things that must be changed if ISEC is to see long range benefits from prototyping or evolutionary development. A major problem for the developers at Fort Lee was the lack of an integrated software tool set to facilitate STAMMIS development. The tools (TAPS and TAPS II) provided by ISEC's Executive Systems Software Directorate (ESSD) are not adequate. In fact, the primary tool used to develop SARSS and ULLS was written in-house at ISEC. More will be discussed about the need for an integrated tool set in chapter 5.

A second potential problem is the ASD's conscious decision that documentation is not an integral part of the evolutionary development process. To adopt such a philosophy is to ignore what history has taught us. Programs that are not documented are more difficult to

understand and therefore maintain. Without documentation, increased maintenance costs may offset the shorter development time that evolutionary development offers. The positive side of this issue is that an integrated software tool set automates much of the documenting process.

G. PROTOTYPING AND EVOLUTIONARY DEVEOPMENT: THE BOTTOM LINE

Traditional analysis methodologies have not adequately come to terms with three overriding and persistent problems of requirements definition. These problems are:

1. Users have extensive difficulties prespecifying final and ultimate requirements;
2. Descriptive and graphic analysis techniques are inadequate to portray the dynamics of an application; and
3. Poor communication is an inherent and debilitating problem among the developer participants.

Well-intentioned efforts to systemize and discipline the process have not solved these problems.

The solution is the evolutionary development of systems by the building and refinement of models. Recent improvements in software tools and fourth generation languages have made evolutionary development both feasible and practical. Evolutionary development should become a key definition strategy for STAMMIS because of the advantages it offers. While not appropriate for all situations, it is a high productivity methodology for solving the requirements definition problem.[8] [Ref. 45: pp. 206-207]

---

[8]Purists would complain that prototyping and evolutionary development are different methodologies and should not be "lumped together." While it may be true that they are different methodologies, they both are similar and useful techniques for solving the specification problems that cripple so many software efforts.

V. THE SOFTWARE DEVELOPMENT ENVIRONMENT

A. INTRODUCTION

Several authors hold a rather narrow view of the software development environment taking it to mean a kind of development system. This author takes a broader view, one similar to Capers Jones:

> The programming environment consists of the sum of the physical facilities, tools, social structures, and intellectual skills dedicated to software production and maintenance by an enterprise [Ref. 13: p. 4].

The physical facilities include such things as office space, small conference rooms for team meetings, and technical libraries. Tools refer to both the hardware and software to support programmers and analysts. Social structures are the formal and informal relationships within an organization. (Organizational policies and programs shape many of the formal and informal structures within an organization. Some of these policies and programs will be discussed in lieu of issues such as the pros and cons of matrix organizations versus functional organizations.) The intellectual skills includes the initial skills employees have when hired and the additional training they receive on the job and in the classroom.

This chapter will evaluate the programming environment at ISEC. A major portion of the chapter will be dedicated to tools which support the programming effort. The investment in the right tools (assuming proper training and use) should provide significant productivity gains. The other environmental factors will be covered, but in less detail. Nevertheless, they are important and must not be ignored.

B. THE PHYSICAL ENVIRONMENT

The physical environment plays a major role in employee motivation, loyalty, and productivity. Many managers assert that people are their most precious asset. If this is true, then employee's work areas should be planned with psychological and physical comforts in mind. This will help eliminate such problems as fatigue, eye strain, and backaches. The proper environment can also improve motivation, increase self-esteem, lessen anxiety levels, and improve concentration. [Ref. 51: pp. 18-19]

The physical environment at ISEC is typical of government office buildings. The facilities were not designed around the specific needs of computer programmers; they were designed for administrative functions. Common complaints at ISEC center on the lack of space, lack of terminals, lack of small rooms for team meetings and walkthroughs, and a temperamental climate control (heating and air conditioning) system. The facilities at ISEC bear little resemblance to IBM's Santa Teresa Laboratory in San Jose, California, which was designed for programming development.

There is no firm agreement as to what constitutes a good physical work space. Opinions offered are highly subjective. [Ref. 52: p. 335] Nevertheless, the design criteria for the Santa Teresa facility might serve as a model or target for which ISEC can aim. Table X presents IBM's primary building and programmer design considerations at the Santa Teresa Laboratory. [Ref. 53: pp. 4-25]

ISEC's present facilities do not measure up well against the industry standard of 90-100 square feet of floor space per employee. Programming requires different types of office space and furnishings than purely administrative functions. The computer printout listings and computer

TABLE X

THE PHYSICAL PROGRAMMER ENVIRONMENT - IBM

BUILDING REQUIREMENTS

- Outside awareness is essential for as many offices as possible. Natural lighting is highly desirable for all work areas.
- Emphasis should be placed on sound proofing, particularly between adjacent offices.
- Maximum flexibility is desired for placement and use of computer terminals and associated work space.
- The site and, in particular, the data processing and project buildings must be secure.

PROGRAMMER REQUIREMENTS

- Communication. The primary consideration in designing the offices is ease of communication. Team members will have to be able to communicate within programming teams, with other teams on the same project, and with other teams world-wide.
- Privacy. Each individual will require a personal work area with an environment that supports the intensive concentration needed for high quality problem solving. Acoustical isolation, adequate ventilation, and individual control of the office environment are key design considerations.
- Furniture. Office furniture and fixtures should be effective for many different tasks. The programmer's basic document is a 15-by-11-inch fanfold program listing which opens to 15 by 22 inches. Work surfaces that can accommodate several listings simultaneously, and lockable storage that can accommodate these documents in hanging vertical files, are required.
- Computer Connections. Every office must have connections to access the computer via video terminals. Preferably, each programmer will have their own terminal.
- Technology. Design flexibility should be maintained with regard to current and future programming technology.

reference manuals consume considerable work and storage space. The programmer needs a desk for manual work, a work

table to spread out listings or notes, a terminal for interaction with the computer, and appropriate storage areas. All this should be encapsulated in a comfortable work environment with adequate lighting, heating, air conditioning, and ventilation. The programming team concept used in large software projects mandates a requirement for small and large conference rooms. These are used for team meetings, walkthroughs, and formal reviews. ISEC is aware of their space constraints and are looking at steps to remedy the situation. The major space problems may be alleviated if ISEC is able to lease additional space in their office building in Falls Church, Virginia (the Melpar Building).

An important question to consider is "how much will productivity increase if we spend X thousand dollars on improvements to the physical environment?" Unfortunately, there are few if any valid, specific studies relating to programming environments. IBM estimates an 11% improvement in productivity at Santa Teresa. They admit that it is nearly impossible to separate gains based on the physical environment and gains based on other factors (e.g. process and technology changes). [Ref. 13: p. 310]

Hertzberg's two-factor motivational model suggests that physical facilities satisfy the hygiene or maintenance needs. This does not mean that upgrading the physical facilities will not yield increases in productivity, efficiency or creativity. Research has shown, though, any increased motivation due to better working conditions may not be as permanent as when the source of someone's motivation is the task itself. [Ref. 54: pp. 12-13] On the other hand, a poor working environment has a negative impact on employee morale, absenteeism, turnover, and productivity. To what degree is very difficult to predict or measure. A survey conducted by Jac Fitz-enz in 1977 revealed that data

processing professionals regard working conditions as less important than employees in other professions. [Ref. 55: p. 126]

C. THE STRUCTURAL ENVIRONMENT

1. Staffing

There is considerable evidence that suggests some programmers are more than an order of magnitude more productive than other programmers. [Ref. 56: p. 846] This implies that management should seek to get these "super programmers" rather than programmers on the low end of the spectrum. ISEC has not attracted this sort of top talent in the past which suggests they can do more in this regard. As a minimum, they can screen potential employees before they enter the intern training program. There are tests available that can be useful for this purpose.

Boehm suggests some staffing principles for software development; Table XI is a summary of his major points. [Ref. 29: pp. 667-672] The principles of job matching and career progression require some discussion. They sound straight-forward and logical but are often not practiced. Some people are placed in a job (e.g. VTAADS maintenance programmer) where they become "irreplaceable" so they get stuck there forever. Other people rise to positions where their technical skills become obsolete after a few years. [Ref. 29: pp. 668-9] Given the rapid evolution in the computer field, we must provide opportunities for employees to grow with the field. The message for management is to keep people motivated by providing an atmosphere where they can fulfil their needs.

Research by Couger and Zawacki has indicated data processing professionals exhibit different motivational tendencies than other workers. In particular, they have a

TABLE XI

STAFFING PRINCIPLES - BOEHM

- Use Better and Fewer People

  The bulk of productivity comes from a relatively small number of participants.

- The Principle of Job Matching

  Fit the task to the skills and motivation of the people available.

- The Principle of Career Progression

  An organization does best in the long-run by helping its people self actualize.

- The Principle of Team Balance

  Select people who will complement and harmonize with each other.

- The Principle of Phaseout

  Keeping a misfit on the team does not help anyone.

higher growth need and lower social need than workers in other professions. [Ref. 57: p. 126] The Fitz-enz study showed that the age and sex of individual programmers and analysts affects how they are motivated. [Ref. 55: p. 127] The lesson for management from this is that they need to learn what motivates their employees to increase productivity.

> With personnel budgets representing an ever-increasing portion of the DP budget, every manager should strive to help the DP staff perform to the best of their abilities. A persistent effort by management can turn motivation from a meaningless buzzword into a valuable tool for improving productivity and reducing turnover. [Ref. 68: p. 9]

2. Awards and Incentives

Various authors have shown that, under the right circumstances, reward systems can contribute to greater employee productivity. Based on the author's on-site visit, this avenue has not explored in any detail at ISEC. One method worth considering is a productivity-based reward system. It is a form of performance bonus system which tie employee earnings to their output. The intent is to motivate employees to produce at an optimum level. A 1980 General Accounting Office (GAO) report states that productivity-based reward systems should not be used in all situations. The report provides the following suggested general principles:

1. Performance should be judged by objective measurable production standards that include all important aspects of the job.

2. The reward offered should be of value to the employee and be significant enough to stimulate effort.

3. The connection between exceeding the production standards and receiving the reward should be clear, and employees should understand the plan.

4. The plan must be accepted by employees and fairly applied by management. [Ref. 59: p. 3]

The report further states that when the above principles cannot be applied, organizations should not attempt to use bonus pay as an incentive for productivity gains. The Office of Personnel Management was to have drafted guidance addressing the design, implementation, and management of a productivity-based reward system. ISEC should attempt to obtain that guidance.

David Sumanth, author of Productivity Engineering And Management, provides a whole chapter on employee-based productivity improvement techniques. One of the techniques

is a group incentive plan called Improshare.[9] Sumanth gives a detailed 3-page explanation of how the plan works (See Sumanth, pages 405-407). Suffice to say, Improshare is designed to share productivity gains between employees and management. No attempt is made to determine the source of the productivity gains or the extent to which each worker contributes. It operates on the premise that workers and management will be interested in improving productivity when both gain something from the increase. [Ref. 22: pp. 405-407] Sumanth offers some other individual and group incentive plans that ISEC may wish to explore (See Sumanth, pages 394-429).

Motivating employees is a management obligation. Seeking new and better ideas to motivate employees to higher levels should be a constant effort. Recognition of outstanding employee efforts is a must for any manager. Awards must be timely and commensurate with the performance. Equally important, but often overlooked, are the day-to-day "strokes" and "pats on the back" which employees deserve for successfully completing assignments and for just "doing their job." Research has indicated that recognition for work completed is perceived as very important to employees. We must never forget that people have feelings; they need to feel wanted and appreciated.

3. Suggestion Programs

Don't all government agencies have suggestion programs? The answer is most have a program, at least in name. But many programs are not active programs. Having a suggestion box up in a few locations can hardly be construed as a bonafide active suggestion program. There are two

---

[9]Improshare is a Registered Service Mark of Mitchell Fein, and is derived from "Improved Productivity Through Sharing."

related techniques which are particularly helpful obtaining and implementing employee ideas. One method involves the establishment of groups of employees who voluntarily cooperate to solve problems related to all facets of a job. These are known as quality circles. They evolved from Japanese management practices and have received considerable press in the last several years. Still, they have been overlooked by most federal agencies.

The second technique is an extension of quality circles called PQ teams. Sumanth coined the term which stands for Productivity and Quality teams. The distinguishing factor between PQ teams and quality circles is that PQ teams are smaller and more functionally specific. Usually a supervisor serves as a team leader to preserve the present authority structure. Also, the group size is normally less than 10 members. [Ref. 22: pp. 421-422] Table XII is a summary of benefits from a PQ team program. Sumanth contends that:

> PQ teams are an effective means of improving employee morale, quality, and productivity in an organization. They have one single purpose in mind: To surface the talents of individuals working in the organization to the maximum extent possible by providing the specialized training and management support necessary to accomplish this.
>
> Team Spirit, positive thinking, and the philosophy of achieving excellence are three important characteristics of PQTs, making them not only efficient in accomplishing improvements in morale, communication, loyalty, productivity, and quality, but also making them effective in achieving organizational goals. [Ref. 22: p. 422]

4. Flextime

Flextime has been a mixed blessing for ISEC. Flextime allows employees to avoid the Washington D.C. rush-hour traffic and take advantage of their "biological

TABLE XII

BENEFITS OF PQ TEAMS - SUMANTH

ORGANIZATIONAL BENEFITS

- Improved product quality and/or service reliability.
- Greater customer satisfaction.
- Reduced costs of operation.
- Greater employee stability.
- More enthusiasm and involvement from employees and management.
- Increased loyalty and commitment to the organization.
- Management can spend more time training employees rather than "ordering" them.
- Improved productivity of operations.

EMPLOYEE BENEFITS

- Greater job security.
- Improved self-image of employees.
- Improved work environment.

clocks." It means employees on flextime arrive fresh rather than tense and "stressed-out." Nevertheless, flextime is only productive if the employees who arrive early (or stay late) make productive use of their time. Managers and programmer team leaders hinted that a good number of flextime employees (20-40%) were not using their "unsupervised hours" efficiently. Reading the newspaper, exchanging the latest gossip, and soaking up several cups of coffee were some typical activities cited by management which are non-productive.

The best way to approach this problem may be simply the installation of a uniformly applied productivity measurement system and good leadership techniques. Flextime is a valuable program which has several benefits. Cancelling the program is not recommended; doing so would create bigger problems.

D. DEVELOPMENT AND MAINTENANCE TOOLS

1. Hardware Considerations

a. Execution Time and Main Storage Constraints

Boehm provides some figures in his book Software Engineering Economics that indicate the procurement of additional computer speed and storage can lead to significant overall systems cost savings. This may be somewhat counter-intuitive but excess capacity can actually save money in the long-run. Boehm says:

> Software productivity can be improved considerably by acquiring enough computer speed and main storage capacity to free the software development from the excess effort required to shoehorn the software within tight execution time and main storage constraints. [Ref. 29: p. 662]

He also provides the following advice:

> 1. Overall system cost is generally minimized by procuring computer hardware with 30 to 50% more capacity than is absolutely necessary.
>
> 2. The more the ratio of software-to-hardware cost increases, the more excess capacity one should procure.
>
> 3. It is far more risky to err by procuring too little hardware capacity than by procuring too much. This is especially important, given the tendencies . . . . for sizing estimates to be low and for software products to expand during development and maintenance. Thus, the preferred amount of hardware capacity procured should be even higher than the minimum system cost level. [Ref. 29: p. 663]

This should become less of a problem as the hardware cost/ performance ratio and storage costs continue to fall.

b. Computer Turnaround Time

The personnel systems and some financial systems STAMMIS are developed and maintained on an IBM 3033 (at the Melpar Building) and on an Amdahl 580 (located at the Vertical Integrated Automation BaseLinE (VIABLE) Regional Data Center (RDC) in Newington, Virginia). There have been complaints that the turnaround time for development and maintenance work is next day service or 24 hours in some cases. This is little better than batch processing. Although measures have been taken to correct this, the programmers and managers still perceive a problem. The procurement of an IBM 3081 at the Melpar Building is underway and may relieve the log jam.

A study was conducted by Major Washburn during a two week individual mobilization assignment annual training period in July, 1984. Washburn suggested that use of micro-to-mainframe hookups to help reduce the turnaround time problem.[10] His idea was to off-load work from the mainframe to some type of programmer work station. Little followup action has been done on Major Washburn's suggestion. This is a sign that the day-to-day workload is so heavy that management does not have time for long-range planning or productivity improvement. Because of the heavy workload, ISEC may want to commission some bright Army graduate school attendees to do thesis research in this area. Specifically, the students could conduct a problem analysis and feasibility study which addresses potential

[10]A good source of information on this topic is a book entitled The Micro-Mainframe Link, published by John Wiley & Sons, Inc.

alternatives and a recommended course of action for ISEC management.

Since Major Washburn's study, there has been some improvement in the products which link mainframe and microcomputers. Many of the products are based on the work station concept. General Dynamics, a leading government contractor, claims to have increased their productivity by 30% using VS COBOL Workbench by Micro Focus, Inc. (of Palo Alto, California). (Their stated productivity goal was 50%.) To obtain their system, General Dynamics used a competitive bid process in which the VS COBOL Workbench was chosen from several bidder's products. It runs on IBM AT and IBM PC/XT hardware which are the type of microcomputers that ISEC has. [Ref. 60: pp. 34-35]

2. Software Tools

a. Introduction

Software tools can radically change and improve the entire software development and maintenance process. A software tool is a computer program designed to automate some portion of the software development and maintenance process. The development tool set should support both management and programmers. Larger software projects usually require a larger proportional period of time and effort for management functions and documentation. In fact, actual coding represents only 15-30% of the total effort on large systems. The remaining time is spent planning, coordinating, communicating, reviewing, testing, and documenting the system.

A 1983 GAO report criticized government agencies for not using software tools during the testing process. Their study revealed that only 13% of the installations surveyed used tools for software testing. [Ref. 61: pp.

12-14] While the GAO study examined only testing, there are many indications that tools are neglected as productivity aides throughout the entire process. A 1984 study of 25 software development environments in the US and Japan surprisingly revealed industry uses software tools sparingly. [Ref. 62: p. 59] Rudy Bazelmans summarized the reasons as follows:

> 1) The hardware engineering background of most managers causes them to (be) unsympathetic to the need for software tools, 2) Most corporations lack an organization whose charter is to evaluate, select, and develop tools, 3) The lack of reuse of tools, 4) The abundance of incomplete or poorly documented tools. [Ref. 63: p. 65]

b. Software Tools - Desirable Features

There are hundreds of software tools available on the market today. Many are fine products, others less so. The tools include text editors, linkers, static analyzers, office automation packages, statistical packages, program management packages, data base management systems, cross-compilers, simulators, emulators, test data generators, test coverage analyzers, application generators, and many others. William Howden provides some insight into the types (and cost) of tools necessary to support large projects such as those ISEC develops. He suggests that the system be built around a software engineering data base which has a version control and automated project control capability. Such a tool system would support all phases of the software life cycle from requirements definition and design to testing and documentation development. [Ref. 64: pp. 321-325] Several authors say that a development software tool set should be able to support these functional areas:

1. Prototyping;

2. Project management and budgeting;
3. Program coding and debugging;
4. Program testing;
5. Data base development; and
6. Automatic generation of development documentation.

To support prototyping, the software tools should be integrated with an active data dictionary. The software and the developer work through the data dictionary so that it maintains a current snapshot or model of the system. [Ref. 45: pp. 118-119] A data base management system (.e.g. Applied Data Research's (ADR) Datacom/DB) to facilitate prototyping and programming functions is very useful. A fourth generation language capability to support prototyping (e.g. ADR's Ideal) is recommended.

Three features are very important in tool procurement. The tools should be compatible - that is, they should be able to communicate with one another without difficulty. They should be easy to use. Tools that are not understood by the programmers will be "left on the self." Tools should support the entire software life cycle. The real key to productivity is automating as much of the process as possible.

c. USE.IT - A New Approach to Systems Development

Most people would agree that automation of the software development process would dramatically increase productivity. Further, if the process could be based on provably correct constructs, testing, system checkout, and maintenance would be sliced to a mere fraction of the effort they now consume. Margret Hamilton and Saydean Zeldin have combined these two features into an integrated family of tools supporting the system life cycle. It is called USE.IT.

One could argue that USE.IT is actually a totally different methodology to approach systems development and does not belong in a section on software tools. The counter argument would be that the basic methodology is the same, only the tools are different. Regardless of where it should be discussed, the concepts behind USE.IT are intriguing. The coeditors of The Journal of Systems and Software had this to say:

> Hamilton and Zeldin's paper on the USE.IT system should be considered "must reading" by all concerned with software development. Their work . . . . has now matured into what James Martin has hailed as the first complete system of tools which can result in provably correct software. It would be surprising, however, if the paper did not evoke controversy among software professionals. [Ref. 65: p. 1]

The paper the coeditors refer to is entitled "The Functional Life Cycle Model and Its Automation: USE.IT." One of the coeditors is Major General Alan B. Salisbury, ISEC's Commander! Below is a brief summary of how USE.IT works.

The first step is to define the requirements. USE.IT has a requirements definition language called AXES. AXES helps users define requirements with either statements or a graphics mode. (AXES is based on Higher Order Software (HOS) Theory which is described in detail in James Martin's text Systems Design From Provably Correct Constructs.) [Ref. 2: pp. 37-143] AXES defines systems from three basic mechanisms: data types, functions, and structures. Axes is not a pure programming language; nor is it a software specification language. It is nonprocedural and can be used to specify systems other than software (e.g. hardware or people systems). [Ref. 36: pp. 40-41]

The next step once the requirements have been defined with AXES is to check the requirements for ambiguity, consistency and completeness. The Analyzer

component of USE.IT performs this task. After any problems identified by the Analyzer are resolved, the requirements are consistent and complete. [Ref. 36: p. 41]

The third step is performed by the Resource Allocation Tool (RAT). From the analyzed AXES specification, the RAT produces code automatically. The RAT will even produce documented code if asked to do so. (Additionally, the AXES front end produces a documented hierarchy of the requirements for the user.) The power and flexibility of the RAT are impressive. [Ref. 36: pp. 41-44]

> The RAT provides the end-user with the capability to reconfigure to any language or machine environment desired, whenever desired, without modifying the requirements definition. Since the Analyzer has guaranteed that the requirements used by the RAT are consistent, the automatic programs produced by the RAT are also consistent. Not only are the initial requirements defined by the user guaranteed to be interface error free after the "programming" phase of development, they are also guaranteed to be the same ones the user defined. [Ref. 36: p. 43]

Another useful feature of the RAT is:

> The same set of requirements that has been "ratted" to one environment (e.g. FORTRAN) can be ratted to another environment (e.g. Ada). This means, for example, that developers who are anxious to start to use the Ada DOD standard language but who do not have the compiler and other support tools yet available can define their requirements in AXES and rat them to FORTRAN or to some other HOL environment until Ada is available. They can simply rat them to Ada when Ada is ready. It also means that developed systems are never obsolete just because there is a new language or a new computer system introduced within an organization. [Ref. 36: p. 43]

The final step includes compilation followed by execution. This is done on what Hamilton and Zeldin call a Higher Order Machine (HOM) which executes the "ratted" requirements. The whole process sounds like a dream come true. Table XIII is a summary of the USE.IT benefits.

TABLE XIII

BENEFITS OF THE USE.IT TOOL SYSTEM

- Interface errors are found automatically before implementation and are eliminated.
- Requirements are complete, consistent and unambiguous.
- Programming is automatic.
- The majority of the documentation can be generated automatically.
- Functional integrity of the requirements is maintained after implementation.
- A means to achieve reuseable software is provided.
- Different requirements definition languages and techniques can be integrated.
- Cost savings of up to 75% can be expected over traditional methods.
- Specifications are easier to modify than with most other techniques.

The predominate use of USE.IT is for designing complex systems. James Martin speculates why:

> Most complex specifications are inadequate. The human mind simply cannot spot the ambiguities, inconsistencies, and incompleteness in highly complex specifications. And a team of human minds is worse because they create pieces that do not mesh exactly. A tool for creating compilable specifications enforces completeness, consistency, and lack of ambiguity in the specifications; otherwise, it cannot generate code for them. [Ref. 2: p. 123]

Martin also says that USE.IT does not eliminate errors in the concept of what a program should do; we can tell it to do something stupid and the methodology can create provably correct code for that stupid function.

USE.IT has been successfully used for large applications. Substantial savings were documented. Hamilton and Zeldin conclude that with USE.IT, an estimated minimum cost savings of 50% results. Perhaps more important, users get what they want because unambiguous requirements definition and rapid prototyping are part of the process.

d. Software Tools at ISEC

The Executive Systems Software Directorate (ESSD) tests and procures software tools for ISEC. The programming directorates have the impression that ESSD could improve the support they provide. Tools such as TAPS are 1960s vintage and are no where near state-of-the-art. Rudy Bazelmans published a recent article entitled "Productivity - The Role of the Tools Group." [Ref. 63: pp. 63-75] He discusses issues which are pertinent to ESSD such as activities the tools group can do to help the programmers.

A significant problem facing ESSD is the long lead time required to evaluate and procure perspective software tools. The economic justification for government procurement is a laborious drill involving many layers of approval. A recommended approach to procuring a unified, integrated tool system is to follow the pattern the Army used on the VIABLE contract. The focus should be on the functions the system must provide rather than on a specific list of hardware and software to accomplish the functions. The system itself should be modular and expandable. A local area network type technology facilitates modular expansion. Once a procurement document is in place, it can be used as a vehicle for future tool system upgrades; the need for re-justification of upgrades should be eliminated. [Ref. 27: pp. 22-26]

In preparing the economic justification for the development tools system, trying to assess how users will use the system is a problem. Determining requirements is no easy task. It will take several months for the programmers to be trained and feel comfortable with the system. Once the become comfortable with the tool system, they will use it in ways that are difficult to anticipate. [Ref. 27: pp. 25-26]

E. THE "INTELLECTUAL SKILLS" ENVIRONMENT

The training program at ISEC is commendable. The courses offered in the continuing education program are particularly outstanding; unless employees attend them, however, they are worthless. Managers and programmer team chiefs must encourage their workers to use this excellent program. They must plan and schedule their employees for appropriate courses. Sending subordinates to training is important for several reasons. Among them are: (1) meeting the high growth needs of DP professionals; (2) keeping current with the latest techniques and technologies; (3) increasing long-run productivity; (4) serving as a reward for hard work or a break from the normal routine; and (5) serving as marketing technique to attract new employees. Managers should also attend certain classes to maintain (or obtain) technical proficiency. The short-run productivity decrease should be offset by increased long-run productivity.

The intern training program at ISEC has been important to the very survival of ISEC. Without it, ISEC's main source of new programmers and analysts would slow to a trickle. Although the intern training program is good, there is some room for improvement. Four criticisms of the program were raised by ISEC employees:

1. Entry screening into the program is too lax;
2. Poor performers in the program are not weeded out;
3. Software testing is not given adequate attention; and
4. There is little "hands on" training with software tools.

These are definite shortcomings. The first two points require management attention. Weak performers are a burden on an organization. Not every person has the mental problem solving abilities that programming requires. The intern program is a good test of one's abilities. Candidates who do not measure up should be phased out. It could be argued that the last two points are better saved for On-the-Job-Training (OJT). This assumes that programmer team chiefs are good teachers and will take the time to adequately train the intern graduates - a risky assumption. The author thinks testing and tools deserve some classroom time even if it means extending the program course length. They are just too important to be left to chance.

F. SUMMARY

This chapter has been a review of the total software development environment. It includes not only the physical working conditions and tools but also the formal and informal organizational relationships and the intellectual skills of the work force. The area requiring the most attention at ISEC is software tools. A unified set of tools that are easy to use, compatible (ability to communicate with each other), and which support the entire software life cycle is needed. The physical working conditions have considerable room for improvement. The training program at ISEC is sound but could use some fine tuning. The structural environment needs some management policy and procedure innovations.

The next chapter focuses on management issues. Management barriers to productivity and software contracts are discussed. Project planning techniques and a software productivity improvement program implementation plan are presented. The chapter also raises some other management issues such as the importance of staying "in touch" customers.

VI. SOFTWARE MANAGEMENT AND PRODUCTIVITY ISSUES

A. INTRODUCTION

Dramatic improvements in hardware and software capabilities during the past several years have steadily increased the complexity of systems development projects. Today's environment requires the close cooperation of technicians, specialists, users, analysts, and programmers. The key to the success of this cooperative effort is effective management procedures, policies, and practices. In fact, of all the variables involved in software development, many experts believe that management is the most important.

This chapter is not about how to manage software development and maintenance. There are many books and articles available for that purpose. The objectives of this chapter are to: (1) discuss some management barriers to productivity; (2) explain common software contract problems and their solutions; (3) discuss project planning systems; and (4) provide the framework for establishing an integrated software productivity improvement program at ISEC.

B. GENERAL MANAGEMENT ISSUES

1. Management Barriers to Productivity

The productivity problem is complex - it is many smaller problems entangled in one large mess. Because it is a complex problem, simple quick-fix solutions are doomed to fail. Sumanth summed up the situation this way:

> Productivity improvement must not be considered as a one shot project or program. It must be on-going and continuous . . . . Whether newspapers or TV make the

> productivity issue a headline story or not, an organization must strive to have a formal productivity process as a normal, routine function. [Ref. 22: p. 477]

Below are seven of the major management barriers to productivity.

a. Short-Run Blinders

There is considerable pressure on all levels of management to look good today so that they can get a good report card based on their short-run results. Despite the lip service paid to the long-run, evaluations are tied to the short-run. Productivity improvements do not happen overnight. Frequently, improvements can decrease productivity in the short-run due to learning curve phenomena. Management must treat productivity as a long term investment. Historically, the biggest gains in productivity have been tied to technology and innovation.

b. The "Desk-Bound and Meeting Syndrome"

Managers tend to spend their time attending meetings, reading and writing staff reports and proposals, and reviewing computer printouts and mounds of paperwork. Unfortunately, the thing that gets neglected is the organization's greatest asset - their people. Productivity suffers because the employees feel ignored and unappreciated.

Two authors in particular are strong advocates of "management by walking around" (MBWA). The term was coined by Tom Peters, co-author of the classic In Search of Excellence. He discusses the concept extensively in another book he co-authored entitled A Passion For Excellence. The other author who favors this management technique is Andrew Grove, president of Intel Corporation (of Santa Clara, California). Here is what Grove says about MBWA:

> There is an especially efficient way to get information much neglected by most managers. That is to visit a particular place in the company and observe what's going on there. Why should you do this? Think of what happens when somebody comes to see a manager in his office. A certain stop-and-start dynamics occurs when the visitor sits down, something socially dictated. While a two-minute kernel of information is exchanged, the meeting often takes a half hour. But if a manager walks through an area and sees a person with whom he has a two minute concern, he can simply stop, cover it, and be on his way. Ditto for the subordinate when he initiates conversation. Accordingly, such visits are an extremely effective and efficient way to transact managerial business. [Ref. 66: p. 49]

Although the concept sounds like apple pie and motherhood, ISEC managers interviewed said they wanted to do more MBWA but just did not have the time. Employees interviewed agreed that they rarely saw mid and top level management "floating" in their work area.

c. Unwillingness to Experiment

Procrastination and constant delays for more research can hamper the initiative and productivity of employees. This is not to say "doing one's homework" is unimportant. It is as long as it does not snuff out enthusiasm and innovativeness. Peters tells us:

> The most important and visible outcropping of the action bias in the excellent companies is their willingness to try things out, to experiment [Ref. 67: p. 134].

d. Ivory Palace Policies

Most authors agree that procedures and standards are key components of the software development and maintenance effort. Nevertheless, some of them are better than others. All too often, these rules and policies are made by the people who sit in their ivory towers. The policy makers do not fully realize the impact on the field

because they have failed to go down "into the trenches" to find out how things really are done and what the real problems are. This suggests that there are too many layers in the organization which isolates top management and increases the communication gap.

e. Lack of Incentives

It is easy to talk about the virtues and necessities of improved productivity. But without any incentives to establish the commitment for improving productivity, it is doubtful there will be much success. The management implications are best summed up in the words of Douglas McGregor "Commitment is a function of the rewards associated with the achievement." As was mentioned in the last chapter, there are several possible individual and group incentive programs that can be used to stimulate commitment.

f. Poor Management Training

Many managers have either too little training or their training is too specialized. What organizations need are managers who are problem solvers, decision-makers, and team builders who can motivate and lead employees. If managers are trained, it is often narrowly focused on such areas as "structured technique number 1" or "operations research technique A." These may be important but not as important as providing adequate leadership and direction to employees.

g. The "Not Enough Time Syndrome"

Several authors suggest that managers will do everything they can to meet some deadline regardless of product quality. It seems there is always time to redo the project but never enough time to do it right the first time.

This was a problem that was raised during the author's visit and will be discussed subsequently.

2. Close to the Customer

The importance of being "close to the customer" in any organization cannot be overemphasized. We discussed this earlier in the chapter on prototyping but it deserves repeating here as a management issue. Peters and Waterman wrote a whole chapter in their book In Search Of Excellence describing the criticality of being "close to the customer." The chapter provides several examples of excellent companies going the extra mile for their customers even when it was not necessarily economical in the short-run. Top management in these companies has made the chain of command understand that service is their business.

> Whether or not they are fanatic in their service obsession as Frito, IBM, or Disney, the excellent companies all seem to have very powerful service themes that pervade the institutions. In fact, one of the most significant conclusions about the excellent companies is that, whether their basic business was metal binding, high technology, or hamburgers, they all defined themselves as service businesses. [Ref. 67: p. 168]

They do this by tailoring their compensation packages, award programs, and training programs so that employees remember how their bread is buttered.

ISEC could learn something from these excellent companies even though ISEC is a non-profit support organization. The functional proponent is actually an ISEC customer. Establishing good working rapport can help eliminate problems such as the FP establishing unrealistic deadlines for projects. This author believes that, where possible, the FP and associated ASD should collocate. This would reduce communication problems and delays. Collocation fosters a sense of team work thus destroying the "We-They Syndrome."

Collocation has been very successful at Fort Lee. One of the most noticeable things during this author's on-site visit was the excellent working relationship that the logistics systems ASD shared with their FP at the Logistics Center (Fort Lee). This was not the case for the other ASDs. The relationship between the FP and ASD affects performance and employee attitudes about their jobs. Because people's motivation plays a significant role in software development, collocation should be carefully considered as a way to increase productivity. It at least deserves further study.

Collocation would entail having the ASD that supports financial systems move to Fort Benjamin Harrison, Indiana. It would also mean that the ASD that supports personnel and force accounting would move to the Military Personnel Center (MILPERCEN) in Alexandria, Virginia (or possibly the Pentagon). The benefits of such a move would have to contrasted with the moving costs. Questions of available space and computer resources need to be addressed.

"Close to the customer" is important in more direct sense as well. Working closely with users in the field is absolutely critical to obtaining complete and accurate specifications when developing new systems. Users help test the system and provide valuable feedback for additional features and corrections. This, in turn, helps reduce total life cycle costs by reducing future maintenance costs, the largest cost driver in most systems. Auerbach Information Management Series provides some additional insight below:

> Organizations have found that the most successful systems are developed with a high degree of user interaction during the design phase. Online systems, for example, which depend heavily on efficient user interactions, are most effective when the user specifies screen designs and functions while the project team advises and performs the technical tasks.
>
> User involvement should be greatest during the initial phases of a development effort, when the systems

requirements are defined in general and in detail. If the user "buys in" at this point, the project's chance for success is greatly enhanced. [Ref. 58: pp. 2-3]

3. System Quality and Productivity

Smart data processing managers are aware that system quality and high productivity are inextricably linked. This may be somewhat counter-intuitive; the effort necessary to achieve quality may lead one to the opposite conclusion. [Ref. 69: pp. 107-108] Generally, the payoffs for "doing it right the first time" are worth the added effort and resources in the long-run. [Ref. 70: p. 115] Recall that errors found during the maintenance phase are several times more expensive to correct than errors found during design.

During the author's visit to ISEC 5-9 June 1985, several programmers and managers were asked: "Given the choice between meeting a deadline with an inferior product or requesting an extension and presenting a polished product, what would ISEC normally chose?" Most people interviewed felt top management would meet the deadline, although some felt this tendency might be changing. The latter is at least encouraging. It does point out a problem that is persistent in many military organizations. Managers tend to take a short-run view of productivity rather than a "We're in this for the long haul" perspective. Quality does not always get the attention it deserves. This was the "No Enough Time Syndrome" mentioned previously in the chapter. Only conscious effort and dedication will turn this situation around.

ISEC is not unlike other government agencies in this regard. Two General Accounting Office (GAO) reports support this. The first, entitled "Federal Agencies' Maintenance Of Computer Programs: Expensive and Undermanaged," explains

problems and guidelines for software maintenance. [Ref. 71: pp. 1-24] The thesis of the second report, entitled "Greater Emphasis On Testing Needed To Make Computer Software More Reliable And Less Costly," is that agencies pay lip service to testing but fail to manage the software testing process. The result is costly, unreliable software. [Ref. 61: pp. 5-14] Both reports are well written and are as applicable today as when they were drafted. ISEC would do well to read these reports and followup on the GAO recommendations. This author's on-site visit flagged these two vital management functions (testing and maintenance) as problem areas at ISEC.

C. SOFTWARE DEVELOPMENT CONTRACTS

The VFDMIS contract has demonstrated, perhaps all too clearly, some of the problems that arise in software development contracts. In 1979, the GAO conducted a study of software development contracts. Specifically, the study included contracts for custom-built applications in the federal government. These are the types of contracts ISEC would use to procure STAMMIS software from a vendor. The report included several causes of problems which were common to all contracts GAO reviewed that encountered difficulties. Table XIV summarizes the GAO findings. [Ref. 72: pp. 1-31]

The difficulties of software development are significant even when the programmers and analysts are from the same organization as the users who need it. Several additional sources of difficulty, as described below, are added when the software is developed by "outsiders." [Ref. 59: pp. 7-8] These include:

1. The problem definition and/or user requirements must be defined so that outsiders can understand it;

TABLE XIV

SOFTWARE DEVELOPMENT CONTRACT PROBLEMS IN THE GOVERNMENT

- Agencies overestimate the stage of systems development they have reached before they contract.
- Contracts fail to stipulate satisfactory performance by the contractor.
- Agencies quickly overcommit themselves and fail to control contractors through strict phasing.
- Agencies do not manage software development contracts during execution.
- Agencies accept and pay for software without adequately inspecting and testing it.
- Contractors claim that agencies fail to provide adequate test data.
- Agencies do not always establish a single focal point for communications with contractors.
- Agencies do not adequately specify or enforce contract clauses for recovery in the event of poor performance by the contractor.
- Contractors frequently fail to provide adequate software documentation.

2. Contracting introduces an extra communication link between the software developer and users;
3. Contractor personnel must be informed about agency operations;
4. Agency management must control the quality of work done outside the agency;
5. First-hand observation of progress is more difficult; and
6. Acquisition of software from a contractor requires an agency to identify and meet all applicable Government procurement regulations.

With all these problems and complicating factors, it seems reasonable to limit software contracts to the absolute minimum possible. Because ISEC will continue to use contracts as an instrument for meeting mission requirements, we need to come to grips with ways to avoid the above problems and difficulties. GAO recommends tapping the resources of the General Services Administration (GSA) and the National Bureau of Standards (NBS) for assistance. This author strongly endorses such a recommendation because there is no need for duplication of efforts. ISEC should benefit by using other government resources. From both a management and taxpayer's perspective, this seems logical.

Besides coordinating with GSA and NBS, the GAO recommends training project managers in the overall skills necessary to manage these contracts. The training should include software engineering, contracting and management. GAO also included a checklist in Appendix I of their report which provides guidance on contracting for software development. [Ref. 59: pp. 29-31]

A recent article in Datamation provides some excellent pointers for solving some of the difficulties enumerated in the GAO report. In the article "Negotiating Software Contracts", author Charles Harris also provides a software contract checklist that appears useful. The article covers various steps that relate to project management, negotiating strategy and other substantive contract issues. Harris' major points are summarized in Table XV below. [Ref. 73: pp. 53-58]

D. PROJECT PLANNING SYSTEMS

1. Preface

Much has been written about project management process techniques. The rampant problems in software

TABLE XV

GUIDELINES FOR SOFTWARE DEVELOPMENT CONTRACTS

NEGOTIATION

- Do not commit yourself too early to a particular vendor or you will lose negotiating leverage.
- Use the negotiation phase to identify and iron out problems and misunderstandings.
- Follow an organized, professional approach to the procurement process to maintain control over the negotiating process.

SPECIFICATION

- Take the time to adequately document specifications.
- Consider a separate consulting agreement for the vendor to develop the specifications at a fixed fee before the software development contract is signed.
- An alternative to a consulting agreement is a staged software development agreement. The user's ability to terminate such a contract is beneficial in encouraging the vendor to produce final specifications that are responsive to user needs.

ACCEPTANCE

- The acceptance procedure may well be the most important user provision in any software acquisition agreement.
- Use a realistic acceptance procedure that tests each module both separately and sequentially.
- Tie payments to deliverables.

WARRANTY, MAINTENANCE and RESTRICTIONS

- To assure performance after acceptance, you need warranty and performance provisions that tailor the vendor's response obligations to your particular needs.
- Be sure the contract describes the scope and response time for all vendor maintenance obligations including routine and critical fixes, enhancements, and upgrades.
- Any restriction of use (e.g. location, machine or site) should be clearly documented.

development have provided many issues to write about! Over the years, we have become smarter about how to manage software development. It is no longer a "shot-in-the-dark guestimate." Nevertheless, it remains complex and difficult. We can and should borrow ideas from other organizations and adapt them to our own systems. We must exercise extreme caution, however, because things like definitions and measures need careful explanation before they can be implemented in a new environment.

Measuring productivity, for example, is not standardized. There is no accepted industry-wide definition of lines of code. What this implies for ISEC is that they should "grow their own" system patterned after those developed elsewhere. There has been some success with project planning systems. Some methods worthy of consideration by ISEC are: (1) an automated computerized project management system; (2) the SLIM system developed by Lawrence Putnam (who was employed at CSC (the forerunner to ISEC) when he did the research in this area!); and (3) the COCOMO system developed by Barry Boehm. They are described briefly in the subsections below. There are several other estimation models

2. Project Management Software

In his book, The Mythical Man-Month, Fred Brooks points out that software projects become late one day at a time. Not all slips are as dangerous as others.

> How does one tell which slips matter? There is no substitute for a PERT chart or critical path schedule. Such a network shows who waits for what. It shows who is on the critical path, where any slip moves the end date.
>
> The preparation of a PERT chart is the most valuable part of its use. Laying out the network, identifying the dependencies, and estimating the legs all force a great deal of very specific planning very early in a project. The first chart is always terrible, and one invents and invents in making the second one. [Ref. 74: p. 49]

One of the problems with PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method) approaches is that they are not updated as changes occur. Many feel that it just is not worth the effort to keep them updated. The result is an obsolete PERT or CPM chart that is no longer of value. To help solve this problem, there are many mainframe and micro-based project management software packages available on the market today.[11]

Mainframe versions such as the Project Tracking System by Management Science America keeps track of the network critical path, organizes detailed accounting information, produces detailed reports of budget variances, and even handles nasty overhead allocations. Micro-based products provide many of the same features but without all the power and depth of mainframe versions. Micro packages are flexible, portable, and relatively inexpensive. They cost between $200 and $10,000 depending on the features and power required.

Gene Schmidt, a consultant, says the key to successful use of project management software is constant checking on the actual progress of the programmers and analysts.

> The tendency of most systems analysts and programmers is not to report productivity problems in the early weeks of a project. They always think that they'll make up the lost productivity next week or the week after. By carefully checking how much work has actually been completed and feeding this information into the computer, we know exactly where we are on a project at any time. [Ref. 75: pp. 46-47]

---

[11]The May 1985 issue of "Software News" contains two articles by Len Horton describing several computerized project management products. He also includes a list of vendors with their addresses.

There are additional benefits to this type of software. The use of project management software would be helpful if a chargeback system is implemented at ISEC.

Certainly, project management software is no panacea. Dr. Francis M. Webster, an expert on project planning software, cautions:

> I have one piece of advice for any manager who is thinking about using project management software . . . . Don't ever use any program to do something you don't understand. If you don't at least understand the calculations that the program is making, you can get burned. [Ref. 76: p. 44]

He adds later, however, that with the proper understanding of project management software, it can be of significant value. As Brooks alluded, it forces a discipline on management. This author recommends ISEC consider and evaluate several available packages for its use.

3. The Putnam SLIM System

The project planning software described above does not really serve as a planning model for project resource estimation but serves more to facilitate the planning process. The Putnam SLIM system, however, is a planning model. Putnam's model is based on the Rayleigh Curve and can be used to estimate the size of medium to large scale software projects. From this basic size estimate, Putnam presents the methodology for converting the size estimate to estimates of time, effort and cost. Putnam suggests the use of simulations and linear programming to assist in the planning process. [Ref. 77: p. 178]

The SLIM system is basically a macro approach to cost and effort estimation that makes reasonable assumptions about the work being done. SLIM is available over timesharing services and could serve as a useful first

approach to developing a project planning model at ISEC. It seems to offer a little less flexibility than you would get by developing your own model, but it is probably a good place to start investigating the subject. [Ref. 27: p. 53] For further information on Putnam's model, the September, October, and November (1979) issues of Datamation provide a 3-part series explaining how to use the model.

4. The COCOMO System

Another project planning model worthy of consideration is the COnstructive COst MOdel (COCOMO) system developed at TRW by Barry Boehm. Boehm provides a detailed description of the COCOMO system in his book Software Engineering Economics. Boehm has 2 versions of COCOMO. The first version is the basic model. It uses the estimated number of lines of code and a variable based on 3 modes of the estimated complexity of the project (roughly easy, medium, or difficult). This is input into an equation that Boehm provides resulting in an estimate of the effort required to develop the software. Unfortunately, the capability of the basic COCOMO system to accurately predict the effort within a factor of 2 is only 60 percent. Some would argue that this is less than impressive.

The second version of the COCOMO system is called the intermediate model. To the basic model, Boehm adds 15 factors or "cost driver attributes." The attributes concern the product itself (e.g. product complexity), computer attributes (e.g. execution time constraints), personnel attributes (e.g. analyst capability), and project attributes (e.g. use of software tools). Each of these parameters are assigned weights (called "effort multipliers" by Boehm), and these weights are used multplicatively to adjust parameters in the model. Boehm claims that with intermediate COCOMO, he is within 20% of the actual results 68% of the time. [Ref. 27: pp. 53-57]

There may be some problems initially testing COCOMO at ISEC. Boehm's data base shows few COBOL data points, therefore, whether his results are statistically valid (for ISEC) is questionable. Further, most of Boehm's data points are for applications other than the development of MIS. Nonetheless, Boehm gives enough detail about how the models are put together that he provides a solid foundation from which to build and calibrate your own project planning system. COCOMO passes what could be called a "reasonable man" test. That is, the parameters in the model are those that an experienced professional would probably expect to find contributing to effort required in a software development project.

With Boehm's model, or any other software development model, one must be careful how it is applied. The development of such an estimation model is an evolutionary process. There are many opportunities for problems. The process is more an art than a science. It will take time to figure out what the bias caused by ISEC measurement practices and how these should be accounted for in the model. As technology changes, the model may require adjustment. It will probably take a number of years to come up with a useful model.[12] [Ref. 27: pp. 57-58]

E. PRODUCTIVITY IMPROVEMENT PROGRAM

It is difficult if not impossible to isolate the ASDs from the their parent organization, ISEC, in terms of improving the total STAMMIS effort. Policies and procedures at ISEC have considerable impact on the process. If we are

---

[12]Graduate students at the Naval Postgraduate School (NPS) are developing a micro-based version of COCOMO using the Knowledgeman data base management system. ISEC may want to contact NPS for further information regarding this research. The point of contact is Professor Tung Bui, Administrative Sciences Department, NPS.

addressing how to improve STAMMIS development and maintenance productivity, we must include the larger system, that is, ISEC.

There are many approaches to productivity improvement at the organizational level. Approaching it in a haphazard way invites problems. The best approach for ISEC may not be the same for another Army organization. Although this author has suggested some possible improvements at ISEC, they must be carefully considered and evaluated. Some involve training costs, others involve purchasing new tools, and still others involve conducting more research (information gathering). Because resources are limited, not all the recommendations can or should be implemented at one time. Tradeoffs must be made. Therefore, some type of mechanism for determining these tradeoffs needs to be established. One such mechanism is the establishment of a software productivity improvement program (SPIP).

Dr. Barry Boehm provides a multi-step process for establishing and tailoring a SPIP in his book Software Engineering Economics. [Ref. 29: pp. 682-689] The following subsections discuss the multi-step process of implementing a SPIP. It is adapted from and based upon Boehm's ideas.

1. OBTAIN TOP MANAGEMENT COMMITMENT

   If managers do not genuinely want improved software productivity, the organization will not get increased software productivity. Managers demonstrate their intent by actions not merely words. They demonstrate their commitment by means of investing in better tools, recognizing and rewarding outstanding performance, and enforcement of standards, etc. Employees are smart. Paying only lip service to productivity improvement will alert employees that productivity is clearly a "back-burner concern."

2. ESTABLISH A PRODUCTIVITY AGENT

An old Army principle says "If you want something to happen, make somebody responsible." The productivity agent serves as the focal point for productivity-related issues. Boehm suggests that software cost estimation and software data collection and analysis activities should be part of the productivity agent's charter. This implies two things. First, productivity duties should be someone's full time job rather than being an additional duty. Second, the productivity agent should have a close connection with the task force for productivity measurement (discussed in chapter 3). To be effective, the productivity agent should report directly to ISEC's Chief of Staff or higher. It may be tempting to put the responsibilities for SPIP on the Quality Assurance Directorate. This may be a good short-term solution while studying possible internal reorganization or awaiting the formal approval of additional spaces to establish a permanent SPIP section, but it is not the recommended long-term solution.

3. ARRANGE BROAD-BASED PARTICIPATION

Implementing a SPIP brings inevitable changes. Allowing the people affected by these changes is good management. It stimulates people's enthusiasm rather than their resistance. It also provides a more accurate assessment of the total environment, Use of productivity officers (discussed in chapter 3) may be a good mechanism to channel employee ideas and suggestions. This would help ensure communication up and down the chain.

4. IDENTIFY OBJECTIVES, ALTERNATIVES, AND CONSTRAINTS

Boehm says the basic objective is to find ways of producing an equivalent level of desired software functionality at a reduced cost, with no loss in product quality while taking care of employees welfare as well. The alternatives to consider are the controllable factors such as use of modern programming practices, use of software tools and improving the work environment. The constraints to consider are government regulations, personnel ceilings, office space limitations, and the hardware and software resources available.

5. EVALUATE ALTERNATIVES AND CHOOSE THE BEST COMBINATION

Boehm emphasizes that by far the best results in productivity improvement are obtained by working the whole problem. There is a potential synergy resulting from integrating a combination of alternatives.

6. PREPARE PHASED IMPLEMENTATION PLAN

The plan should be incremental with the early phases concentrating on the more straight forward, easy-to-implement, high payoff items. Like all good plans, it must address the "why", "what", "where", "who", "when", "how", and "how much" questions.

7. OBTAIN THE AUTHORITY TO PROCEED

The additional resources required to implement the plan for new or increased salaries, equipment, tools, training, etc., require the authority to commit funds for their procurement. These must be considered as an investment. Productivity is not free. One should

not expect large increases without some resources to achieve them. The farmer who uses a tractor will be much more productive than one behind an ox.

8. IMPLEMENT THE WHOLE PLAN

Not all parts of a productivity plan may be exciting to implement. Getting rid of employees that are holding back progress is not a pleasant task but if it is not done, the ill effects on other employees will cause further damage. Thus, the whole plan needs to be implemented, not just the fun parts.

9. FOLLOWUP AND ITERATE

The followup implies that ISEC has a measurement system in place. Without it, there is no mechanism other than "gut feel" to gauge productivity. To assist in the measurement problem, Boehm has a series of forms that may be helpful in tracking projects. These are included in Appendix A of his book, Software Engineering Economics. No long-range plan is perfect. What is good today may not be so tomorrow. The "iterate" portion of this step allows for the fact that the SPIP is evolutionary and should be changed as needed.

Boehm, in another article that he co-authored, makes some additional points about SPIP and productivity improvement that are important. Table XVI summarizes these points. Some are redundant with material presented earlier but are worth repeating. [Ref. 3: pp. 30-42]

Finally, Boehm reminds us that improved software productivity is not an end in itself. It is a means to help people better expand their capabilities to deal with data, information, and decisions. He emphasizes that the software

TABLE XVI

KEY POINTS - SOFTWARE PRODUCTIVITY IMPROVEMENT PROGRAM

- Significant software productivity improvements are not achieveable without the full commitment of higher management.
- The best way to get started on a sustained SPIP is to establish a software productivity agent.
- Significant productivity gains require an integrated program of initiatives in several areas.
- An integrated can have an extremely large payoff.
- Improving software productivity involves a long, sustained effort.
- In the very long-run, the biggest productivity gains will come from increasing the use of existing software (tools and utilities suitable for reuse).

productivity scoreboard is just one of many ways we have to gauge our progress toward becoming more efficient data processing professionals. [Ref. 29: p. 689]

Besides the practical advice of Dr. Boehm on SPIPs, Sumanth provides some useful information on (generic) productivity improvement programs. Condensed in Table XVII are common problems with productivity improvement programs and how to avoid these pitfalls. [Ref. 22: pp. 483-486]

F. SUMMARY

This chapter has discussed a number of relevant management barriers to productivity. Contract management problems and solutions were identified. Project planning systems were explained. A recommendation was made that ISEC should begin preliminary work on developing a project

TABLE XVII

PRODUCTIVITY IMPROVEMENT PROGRAM PROBLEMS AND SOLUTIONS

Resistance to Change

- Get employees involved. Establish appropriate financial and non-financial incentives.

Inadequate Planning

- Carefully plan the program and its implementation. Brainstorm possible problems and develop prevention strategies.

Modification in Data Collection

- Develop a pilot system. Test this system with hypothetical data.

"It's Not My Program Syndrome"

- Consult employees before the program is established for their ideas and suggestions. Make the employees a part of the program.

Misinterpretation and Misuse of the Results

- Carefully evaluate the results. Don't read more into the results than is actually there. Look at the big picture. Ensure incentives do not encourage employees to suboptimize some aspect to the detriment of the total process.

Unwillingness to Share Productivity Gains

- Recognize employees that contribute to the effectiveness of the program. Without any incentives, long-term commitment to the program will be threatened.

Tendency to Compromise Quality for Productivity

- Managers and employees must be trained that quality and productivity are related. Both are important.

planning and effort estimation model. The framework for establishing a software productivity improvement program at ISEC was outlined and the benefits and potential problems of such a program given. This author recommends ISEC initiate

the planning for integrated software productivity program soon. Although there are definite short-term expenses to be shouldered, the long-run payoff can be extremely high.

The next and final chapter provides the author's conclusions and recommendations for productivity improvement at ISEC with particular emphasis on the STAMMIS development and maintenance process. It also provides recommendations for areas requiring additional research.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. THESIS SUMMARY

This is a report of the productivity enhancement study of the ISEC STAMMIS development and maintenance effort. The study is an initial effort to identify candidate practices, areas and projects for productivity improvement. We have reviewed the STAMMIS development and maintenance process at ISEC and have identified problem areas. To solve these problems and improve productivity at the programmer, project and organizational levels, the entire process was examined for candidates for improvement. Four major areas were suggested for improvement: methodology, management, technology, and the software development environment.

Under the methodology umbrella, problems with the traditional software life cycle and specification process were emphasized. An alternative life cycle using evolutionary development was suggested. The advantages and limitations of prototyping were discussed. In addition, the applicablity of prototyping at ISEC was established. Probably the most important contribution of prototyping and evolutionary development is that they tear down the walls of misunderstanding and miscommunication thus bringing users and developers together.

Several management issues were raised. Problems with software contract management were explored. Some suggestions for prevention of these problems were offered. Helpful hints about the negotiation of contracts were also provided. Project planning was discussed and a few possible approaches were presented. The need for a productivity improvement program at an organizational level was established.

The total software development and maintenance environment was considered. To assist in prototyping and structured systems development, a unified software tool set is a necessity. At the heart of this tool set is an integrated and active data dictionary. It provides a basis for a record management system and automates much of the documentation process. The environment also includes the human factors involved in the software process. These are such things as the physical work space, the training program, the awards and incentives program, compensation, etc. Several of these are candidates for improvement at ISEC.

Productivity was defined and distinguished from production, efficiency, and effectiveness. The problems associated with measuring performance was contrasted with the handsome benefits that measurement offers. Some guidelines were included to aid in the proper evaluation of productivity measures. Several productivity measures for software development and maintenance were explained and evaluated. Finally, a strategy for implementing a productivity measurement system at ISEC was presented.

B. RECOMMENDATIONS

To be as useful and realistic as possible, the study included five days of on-site interviews and observations in addition to the review of several dozen working documents at ISEC and over one hundred articles and books related to this study. Although a longer on-site period followed by a second visit to probe some areas more thoroughly would have been optimal, the results should be valuable to ISEC. They provide an outsider's view of problems and potential solutions relating to the entire software process.

The major recommendations (in relative order of importance) in this report are described below.

1. ISEC should seek a unified software tool set that will support all facets of software development and maintenance. The tools chosen should support prototyping and communicate with each other.
2. Prototyping/Evolutionary development should be adopted as the preferred methodology for requirements specification and systems development. They reduce project risk, yield systems which better reflect user needs, and usually reduce development time and manpower requirements.
3. The implementation of a productivity measurement system will help ISEC's management in planning, controlling, and evaluating the software development and maintenance process.
4. ISEC software contract management needs revamping. ISEC needs to train its management in appropriate contracting areas.
5. ISEC should seek the help of other federal agencies (such as the GAO, GSA, and NBS), and DOD organizations (such as the National Security Agency, Air Force and Navy) to improve all aspects of the software process. There is no need to reinvent the wheel; a suitable framework, at least, probably exists at one of these organizations.
6. The physical facilities at ISEC are far from ideal. In need of improvement are the amount of work space per employee, the number of terminals, the computer response time, the number of conference rooms for reviews and walkthroughs, use of office automation, and the heating/air conditioning system.

The above suggestions should be incorporated into a total software productivity improvement program (SPIP). The SPIP

would include a clear statement of organizational productivity objectives and a carefully thought-out plan to achieve them. The SPIP might include some promising productivity features such as tailoring a library of reusable code or creating an information center to support end-users. Results in some commercial organizations demonstrate that productivity could double within two years with a SPIP and the long-run gains may be even more spectacular. [Ref. 3: p. 33] Organizational commitment is critical to the success of the program. It should start with top management and echo through all levels of the organization.

C. RECOMMENDATIONS FOR FURTHER STUDY

During the author's on-site visit and literature review, several intriguing issues surfaced that may deserve study or further analysis. These issues/areas for further study (in no particular order) are discussed below.

1. Conduct a detailed analysis of software tools applicable for ISEC addressing such issues as "What is currently on the market?", "What do ISEC managers and programmers want?", "What can ISEC afford?" and "How can ISEC speed up their software tools acquisition and training program?".
2. Conduct a feasibility study identifying and quantifying the costs and benefits of the Army adopting a single operating system such as MVS for STAMMIS.
3. Conduct a capacity planning analysis of the computer resources available for STAMMIS development and maintenance.
4. Participate in the preliminary planning and requirements analysis for the establishment of STAMMIS corporate data base.

5. Although DOD and General Paige seem committed to Ada, a detailed cost/benefit analysis of it might be enlightening.
6. Conduct and analyze a series of employee surveys regarding: (1) their perception of problems at ISEC; (2) their ideas and suggestions for improving productivity; and (3) what motivates them.
7. Conduct an analysis concerning the effect of a chargeback system (.e.g. billing DCSPER for SIDPERS maintenance work) on STAMMIS functional proponents, user demand and user perceptions. The study might include the best criteria to bill the FPs and propose an implementation plan for installing such a chargeback system.
8. Conduct a review and analysis of good ideas already in practice in other federal and DOD organizations concerning: (1) contract management; (2) productivity measurement; and (3) productivity improvement programs; (4) software development environments; (5) awards and incentive programs; and (6) software tools in use.
9. Conduct a feasibility study for the collocation of all ISEC programming directorates with their FP counterparts.
10. Conduct research to determine what project planning system would be most useful for ISEC. Micro-based systems, the Putnam SLIM system, and the COCOMO model are some potential candidates for consideration at ISEC.

APPENDIX A
ACRONYMS

ACSIM	- Assistant Chief of Staff for Information Management
AESEIC	- U.S. Army Electronics System Engineering and Installation Command
ASD	- Assigned System Developer
COA	- Comptroller of the Army
COBOL	- COmmon Business Oriented Language
CPM	- Critical Path Method
CPO	- Civilian Personnel Office
CSC	- U.S. Army Computer Systems Command
DA	- Department of the Army
DCSIM	- Deputy Chief of Staff for Information Management
DCSLOG	- Deputy Chief of Staff for LOGistics
DCSPER	- Deputy Chief of Staff for PERsonnel
DOD	- Department of Defense
ESSD	- Executive Systems Software Directorate
FORTRAN	- FORmula TRANslation
FP	- Functional Proponent
GAO	- General Accounting Office

GSA	- General Services Administration
HOL	- Higher Order Language
HOM	- Higher Order Machine
HOS	- Higher Order Software
IBM	- International Business Machines, Inc.
IE	- Information Engineering
IMA	- Information Mission Area
IRM	- Information Resource Management
ISEC	- U.S. Army Information Systems Engineering Command
ISSSC	- U.S Army Information Systems Software Support Command
MILPERCEN	- MILitery PERsonnel CENter
MIS	- Management Information Systems
MVS	- Multiple Virtual System
NBS	- National Bureau of Standards
NPS	- Naval Postgraduate School
NSA	- National Security Agency
OMB	- Office of Management and Budget
OJT	- On-the-Job-Training
PC	- Personal Computer
PERT	- Program Evaluation and Review Technique
PQ Teams	- Productivity and Quality Teams
RAT	- Resource Allocation Tool

RDC - Regional Data Center

SAILS - Standard Army Installation Logistics System

SARSS - Standard Army Retail Supply System

SDLC - Systems Development Life Cycle

SIDPERS - Standard Installation/Division PERsonnel System

SPIP - Software Productivity Improvement Program

STAMMIS - Standard Army Multi-command Management Information System

STANFINS - STANdard FINance Systems

STARCIPS - STandard ARmy CIvilian Payroll System

STEP-UP - Systems Through an Evolutionary Process Using Prototyping

ULLS - Unit Level Logistics System

USA - United States Army

USACC - U.S. Army Communications Command

USAISC - U.S. Army Information Systems Command

VIABLE - Vertical Integrated Automation BaseLinE

VFDMIS - Vertical Force Development Management Information System

VTAADS - Vertical The Army Authorization Document System

APPENDIX B

USING GRADUATE STUDENTS TO IMPROVE PRODUCTIVITY

This author contends that Army organizations such as ISEC should actively tap the pool of talent of Army fully-funded advanced degree students (and possibly other Armed Forces students) to the maximum extent possible. Most students must complete a thesis to fulfil their degree requirements. Many have no idea what they would like to do their research on. It makes economic sense to obtain a return on the Army's investment in their education (in addition to their service obligation).

To actively recruit students for thesis work will require 3 things.

1. ISEC will need to identify target projects in "thesis-sized chunks". Large projects will have be logically decomposed.
2. ISEC will need to budget TDY funds for research expenses. Typically, expenses for a thesis range from $1000 to $3000. There are many variables involved but a good planning figure would be $2000 per study.
3. ISEC should send knowledgeable representatives who relate well to students to market their theses research topics. Timing of these visits is critical to the success of the program. Too late and everyone already has a topic; too early and no one will know what they want to do.

Remember that each student has at least one and usually two PhDs for advisors so the quality of the product is generally good. Using Army students to solve Army problems helps everyone. It is a viable alternative to using

consultants or in-house assets. To ignore the student knowledge-base is to waste an Army resource.

Besides the students, the faculty at many educational institutions are always looking for interesting research work. They have a special expertise worth considering for those projects which ISEC cannot (for whatever reason) do themselves or which are beyond the scope of student thesis work.

LIST OF REFERENCES

1. Moore, Barry M., "Systems and Application Development", The Economics of Information Processing, vol. 2, John Wiley & Sons, Inc., 1982, pp. 83-88.

2. Martin, James, Systems Design From Provably Correct Constructs, Prentice-Hall, Inc., Inglewood Cliffs, New Jersey, 1985.

3. Boehm, Barry W., Dr., et al., "A Software Development Environment for Improving Productivity", IEEE Computer, vol 17, no. 6, June 1984, pp. 30-44.

4. Martin, James, Application Development Without Programmers, Prentice-Hall, Inc., Inglewood Cliffs, New Jersey, 1982.

5. Booch, Grady, Software Engineering With ADA, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1983.

6. Doyle, David K., Lieutenant General, USA and Craven, Ronald E., "Mission: To Bring Order To Information Revolution", Army, July 1985, pp. 30-33.

7. "Army Seeks Hill O.K. for Fifth Deputy Chief of Staff", Army Times, 17 September 1984, p. 4.

8. Donahue, Robert J., Major General, USA and Craven, Ronald E., "Directorates Spearhead Merger Initiative", Army, July 1985, pp. 33-35.

9. "Belvoir Command Renamed", Army Times, 12 August 1985, p. 38.

10. Nolan, Richard L., "Managing the Crises in Data Processing", Harvard Business Review, vol. 57, March--April 1979, pp. 115-126.

11. Senn, James A., Information Systems in Management, Second Edition, Wadsworth Publishing Company, 1982.

12. "Army Automation Software Design and Development", Technical Bulletin 18-103, Headquarters, Department of the Army, Washington D.C., January, 1983.

13. Jones, Capers (ed.), Programmer Productivity: Issues for the Eighties, IEEE Catalog No. EHO 186-7, IEEE Computer Society, P.O. Box 80452, Los Angles, California, 1981.

14. U.S. Army Information Systems Software Support Command, Interview with managers, Personnel and Force Accounting Directorate, during author visit, 7 June 1985.

15. Edwards, Mark H., "Software Salaries Survey - Where Do You Stand?", Software News, July 1985, pp. 58-69.

16. Cozette, Chuck, "27th Annual DP Salary Survey - Prosperity Continues", Infosystems, June 1985, pp. 27-38.

17. U.S. Army Information Systems Software Support Command, Interview with VFDMIS managers, Personnel and Force Accounting Directorate, during author visit, 8 June 1985.

18. Kroenke, David, Database Processing, Second Edition, Science Research Associates, 1983.

19. U.S. Army Computer Systems Command, Letter, Subject: New Approaches for Systems Design and Maintenance Techniques, unclassified, 20 July 1984.

20. U.S. Army Communications Command, untitled letter (with enclosure), unclassified, 21 September 1984.

21. Siegel, Irving H.,Dr., Company Productivity, The W. E. Upjohn Institute for Employment Research, Kalamazoo, Michigan, 1980.

22. Sumanth, David J.,Dr., Productivity Engineering and Management, McGraw-Hill Book Company, Inc., 1984.

23. Toffler, Alvin, Previews and Premises, William Morrow & Company, New York, 1983.

24. Parikh, Girish, "Techniques of Measuring Programmer Productivity", Data Management, June 1985, pp. 18-29.

25. Jones, T. C., "Measuring Programming Quality and Productivity," IBM Systems Journal, vol. 17, no. 1, 1978, pp. 39-63.

26. Boehm, Barry W., Dr.,, "Software and Its Impact: A Quantitative Assessment", Datamation, May 1973, pp. 48-59.

27. Lyons, Norman R., and Boger, Dan C., A Productivity Enhancement Study of the FMSO Software Effort, November, 1983.

28. Arthur, Lowell Jay, Programmer Productivity, John Wiley & Sons, Inc., 1983.

29. Boehm, Barry W., Dr., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

30. Crossman, Trevor D., "Taking the Measure of Programmer Productivity", Datamation, May 1979, pp. 144-147.

31. Drummond, Steve, "Measuring Applications Development Performance", Datamation, 15 February 1985, pp. 102-108.

32. Brown, Patrick, "Managing Software Development", Datamation, 15 April 1985, pp. 133-136.

33. Fairley, Richard E. Software Engineering Concepts, McGraw-Hill, Inc., 1985.

34. Gilb, Tom, Software Metrics, Winthrop Publishers, Inc., Cambridge, Massachusetts, 1977.

35. U.S. Army Information Systems Software Support Command, Slide presentation and briefing by managers, Development Center Ft Lee, during author visit, 6 June 1985.

36. Hamilton, M., and Zeldin, S., "The Functional Life Cycle Model and Its Automation: USE.IT", The Journal of Systems and Software, vol. 3, no. 1, March 1983, pp. 25-62.

37. Bunyard, Jerry M., Major General, USA and Coward, James M., "Today's Risks in Software Development - Can They Be Significantly Reduced?", Concepts - The Journal of Defense Systems Acquisition, vol. 5, no. 4, Autumn 1982.

38. McCracken, David D. and Jackson, Michael A, "Life-Cycle Concept Considered Harmful", ACM Software Engineering Notes, vol. 7, no. 2, April 1982, pp. 29-32.

39. Gladden, G. R., "Stop the Life-Cycle, I Want to Get Off", ACM Software Engineering Notes, vol. 7, no. 2, April 1982, pp. 35-39.

40. Boehm, Barry W., Dr., "Seven Basic Principles Of Software Engineering", The Journal of Systems and Software, vol. 3, no. 1, March 1983, pp. 3-24.

41. Paige, Emmett., Lieutenant General, USA, Speech, "Army's New Approach to Information Systems Acquisition", Washington D.C., 12 April 1985.

42. "Prototyping: Modeling Your Way Through", Software News, vol. 4, no. 8, August 1984, p. 29.

43. Jones, Clifford B., Software Development, Prentice-Hall International, London, 1980.

44. Brooks, Frederick P. Jr., The Mythical Man-Month, Addison-Wesley Publishing Company, Reading, Massachusetts, 1975.

45. Boar, Bernard H., Application Prototyping, John Wiley & Sons, Inc., 1984.

46. Noel, Alan F., Major, USA, Prototyping with Data Dictionaries for Requirements Analysis, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1985.

47. "Speeding Up Application Development", EDP Analyzer, vol. 23, no. 4, April 1985, pp. 1-16.

48. Parnas, David L., "Designing Software for Ease of Extension and Contraction", IEEE Transactions on Software Engineering, March 1979, pp. 226-235.

49. Orr, Kenneth, "Managing the Software Crises", Computerworld, 15 July 1985, pp. ID/1-ID/9.

50. Powers, James H., Colonel, USA, Unpublished Document, "STEP-UP - Getting Better Computer Systems to the Army Faster", undated.

51. Goodwin, Connie, "Considerate, Carefully Planned Office Environments Reduce Employee Anxiety", Data Management, June 1985, pp. 18-19.

52. Wasserman, Anthony I., "Organizational and Ergonomic Aspects of Software Development Environments", Software Development Environments, IEEE Catalog No. EHO 187-5, IEEE Computer Society, Los Alimitos, California, 1981, pp. 335-6.

53. McCue, Gerald M., "IBM's Santa Teresa Laboratory - Architectural Design for Program Development", IBM Systems Journal, vol. 17, no. 1, 1978, pp. 4-25.

54. Whitacker, Richard M., Application of Motivational Models, Master's Thesis, University of Virginia, Fairfax, Virginia, August 1982.

55. Fitz-enz, Jac, "Who is the DP Professional", Datamation, September 1978, pp. 124-128.

56. Curtis, Bill, "Substantiating Programmer Variability", Proceedings of the IEEE, vol. 69, no. 7, July 1981, p. 846.

57. Couger, Daniel J., and Zawacki, Robert A., "What Motivates DP Professionals", Datamation, September 1978, pp. 116-123.

58. Stanley, Frank, J., "Motivating DP Personnel", Auerbach Information Management Series, no. 2-01-06, Auerbach Publishers Inc., Philadelphia, Pennsylvania, September/October 1982.

59. "Ways To Improve Federal Management And Use Of Productivity Based Reward Systems", GAO Report To The Congress of the United States, no. FPCD-81-24, 31 December 1980.

60. Schindler, Paul E., "General Dynamics' MIS Finds Way To Raise Productivity 50%" Information Week, 29 July 1985, pp. 30-35.

61. "Greater Emphasis On Testing Needed To Make Computer Systems More Reliable And Less Costly", GAO Report To The Congress of the United States, no. GAO-IMTEC-84-2, 27 October 1983.

62. Zelkowitz, Marvin V., et al., "Software Engineering Practices in the US and Japan", IEEE Computer, vol. 17, no. 6, June 1984, pp. 57-66.

63. Bazelmans, Rudy, "Productivity - The Role of the Tools Group", ACM SIGSOFT Software Engineering Notes, vol. 10, no. 3, July 1985, pp. 63-75.

64. Howden, William E., "Contemporary Software Development Environments", Communications of the ACM, vol. 25, no. 5, May 1982, pp. 318-329.

65. Salisbury, Alan B., and Manley, John H., "Editor's Introduction", The Journal of Systems and Software, vol. 3, no. 1, March 1983, p. 1.

66. Grove, Andrew S., High Output Management, Random House, Inc., New York, 1983.

67. Peters, Thomas J., and Waterman, Robert H. Jr., In Search of Excellence, Warner Books, Inc., New York, 1982.

68. Stanley, Frank J., "Establishing a Project Management Methodology", Auerbach Information Management Series, no. 3-10-01, Auerbach Publishers Inc., Philadelphia, Pennsylvania, undated.

69. Inmon, William H., Management Control of Data Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

70. Arthur, Jay, "Software Quality Measurement", Datamation, 15 December 1984, pp. 115-120.

71. "Federal Agencies' Maintenance Of Computer Programs: Expensive And Undermanaged", GAO Report To The Congress of the United States, no. AFMD-81-25, 26 February 1981.

72. "Contracting For Computer Software Development - Serious Problems Require Management Attention To Avoid Wasting Additional Millions", GAO Report To The Congress of the United States, no. FGMSD-80-4, 9 November 1979.

73. Harris, Charles E., "Negotiating Software Contracts", Datamation, 15 July 1985, pp. 52-58.

74. Brooks, Frederick P. Jr., "The Mythical Man-Month", Datamation, December 1974, p. 44-52.

75. Horton, Len, "Users Find a Tool To Manage Themselves", Software News, May, 1985, pp. 45-49.

76. Horton, Len, "Could Project Management Be The Next 'Super Product'?", Software News, May, 1985, pp. 41-44.

77. Putnam, Lawrence H., and Fitzsimmons, Ann, "Estimating Software Costs", Datamation, October, 1979, pp. 171-178.

INITIAL DISTRIBUTION LIST

No.		Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey California 93943-5100	2
3.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey California 93943-5100	1
4.	Professor David R. Whipple Code 54wp Department of Administrative Sciences Naval Postgraduate School Monterey California 93943-5100	1
5.	Associate Professor Norman R. Lyons Code 541b Department of Administrative Sciences Naval Postgraduate School Monterey California 93943-5100	1
6.	LTC Richard E. Broome 6405 Wynegate Drive Springfield, Virginia 22152	2
7.	CPT Timothy F. Robertson 1147 St. Paul Avenue St. Paul, Minnesota 55116	7
8.	Professor Gordon H. Bradley Code 52bz Department of Computer Science Naval Postgraduate School Monterey California 93943-5100	2
9.	Associate Professor Michael P. Spencer Code 54xq Department of Administrative Sciences Naval Postgraduate School Monterey California 93943-5100	1
10.	Commander U.S. Army Information Systems Engineering Command Ft Belvoir, Virginia 22060-5456	10

C 05 - 197

Thesis
R5997
c.1

Thesis
R5997
c.1

Robertson
A productivity enhancement study for the U.S. Army Information Systems Engineering Command.

22 JUN 87 31893
25 SEP 87 31893
31 MAY 89 60828
4 MAY 90 36528

Thesis
R5997
c.1

Robertson
A productivity enhancement study for the U.S. Army Information Systems Engineering Command.

thesR5997
A productivity enhancement study for the
3 2768 000 68443 5
DUDLEY KNOX LIBRARY